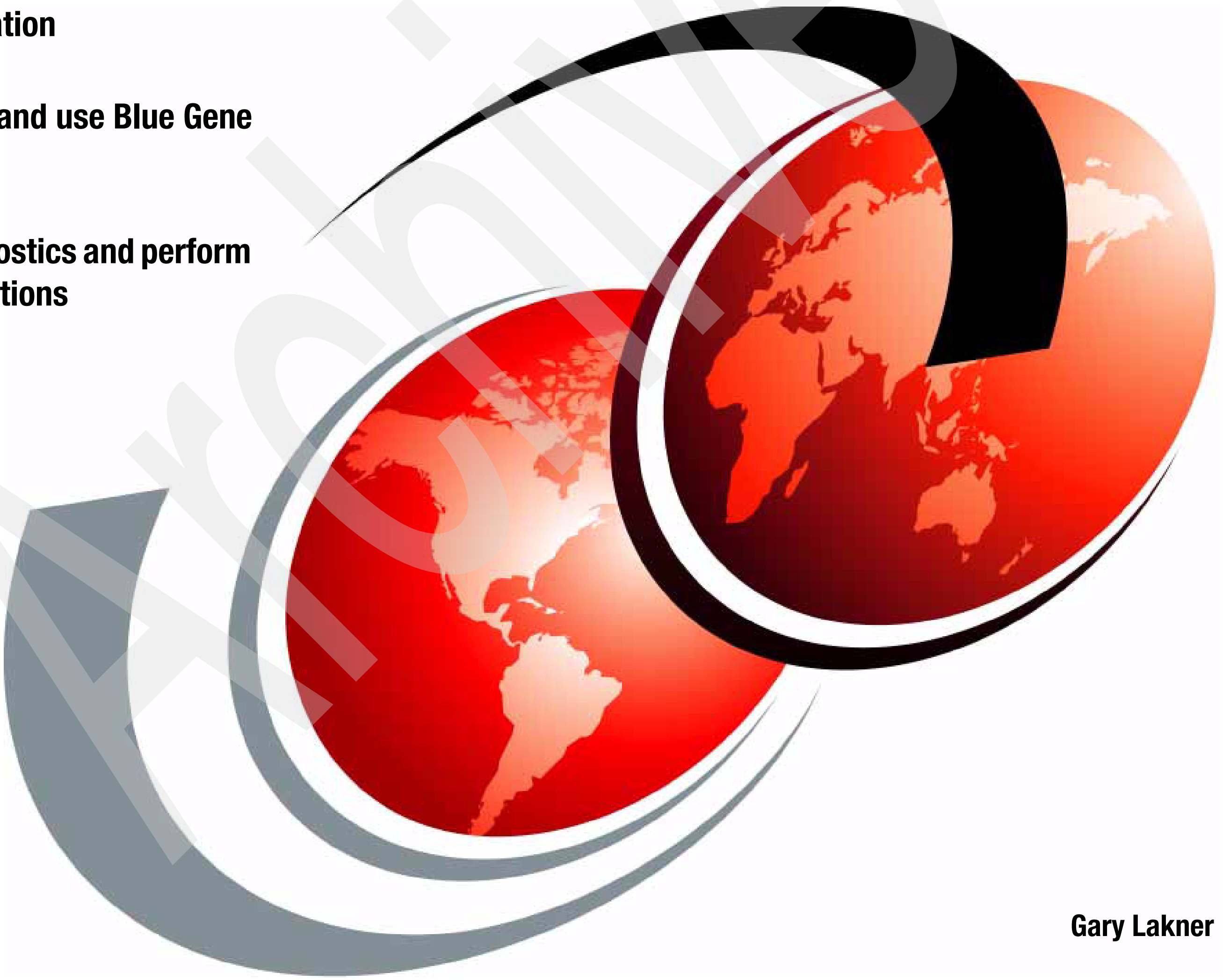
IBM System Blue Gene Solution: Blue Gene/P System Administration

Perform Blue Gene/P system administration

Configure and use Blue Gene Navigator

Run Diagnostics and perform Service Actions



Gary Lakner

Redbooks

ibm.com/redbooks

International Technical Support Organization

IBM System Blue Gene Solution: Blue Gene/P System Administration

September 2009

SG24-7417-03

Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

Fourth Edition (September 2009)

This edition applies to Version1, Release 4, Modification 0 of Blue Gene/P (product number 5733-BGP).

Contents

Notices

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at:

http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

Blue Gene/L™
Blue Gene/P™
Blue Gene®
DB2®
eServer™
GPFS™
IBM®
LoadLeveler®
PowerPC®
Redbooks®
Redbooks (logo) ®
System i®

The following terms are trademarks of other companies:

Snapshot, BareMetal, and the NetApp logo are trademarks or registered trademarks of NetApp, Inc. in the U.S. and other countries.

SUSE, the Novell logo, and the N logo are registered trademarks of Novell, Inc. in the United States and other countries.

Red Hat, and the Shadowman logo are trademarks or registered trademarks of Red Hat, Inc. in the U.S. and other countries.

Java, JDBC, Power Management, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Internet Explorer, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

This IBM® Redbooks® publication is one in a series of books that are written specifically for the IBM System Blue Gene® supercomputer, Blue Gene/P™, which is the second generation of massively parallel supercomputer from IBM in the Blue Gene series. It provides an overview of the system administration environment for Blue Gene/P. It is intended to help administrators understand the tools that are available to maintain this system.

This book explains briefly the evolution of the system, from Blue Gene/L to the new Blue Gene/P. It details Blue Gene Navigator, which has grown to be a full featured Web-based system administration tool on Blue Gene/P. The book also describes many of the day-to-day administrative functions, such as running Diagnostics, performing Service Actions, and monitoring hardware. There are also sections to cover BGPMaster and the processes that it monitors. The final chapters of the book cover the tools that are shipped with the system and details about configuring communications with the I/O nodes and Power Management™ features.

The team who wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Poughkeepsie Center.

Gary Lakner is a Staff Software Engineer for IBM Rochester on assignment in the ITSO. He is a member of the Blue Gene Support Team in the IBM Rochester Support Center, where he specializes in both Blue Gene hardware and software, as well as performs customer installations. Prior to joining the Blue Gene/L™ team, he supported TCP/IP communications on the IBM eServer™ System i® server. Gary has been with IBM since 1998.

Thanks to the following people for their contributions to this project:

Paul Allen
John Attinella
Mike Blocksome
Lynn Boger
Kathy Cebell
Todd Inglett
Tom Gooding
Nicholas Goracke
Tom Liebsch
Cory Lappi
Chris Marroquin
Sam Miller
Mark Megerian
Sam Miller
Mike Mundy
Roy Musselman
Tom Musta
Mike Nelson
John Orbeck
Jim Van Oosten
Jeff Parker

Ruth Poole
Joan Rabe
Joseph Ratterman
Don Reed
Harold Rodakowski
Richard Shok
Brian Smith
Will Stockdell
Andrew Tauferner
Kiswanto Thayib
Bin Ye
IBM Rochester

Special thanks to Brant Knudson and Tom Budnik of IBM Rochester for their time and contributions to the project.

Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Summary of changes

This section describes the technical changes made in this edition of the book and in previous editions. This edition might also include minor corrections and editorial changes that we do not identify here.

Summary of Changes
for SG24-7417-03
for *IBM System Blue Gene Solution: Blue Gene/P System Administration*
as created or updated on August 6, 2010

September 2009, Fourth Edition

This revision reflects the addition, deletion, or modification of new and changed information described in the following sections.

New information

- Added configuration and usage information about the Event Log Analysis tool in 3.7, "Event Log Analysis" on page 97
- Added authentication information to 6.2.1, "Submit client to submit mux" on page 139
- Described the new Event Log Analysis function that was added in V1R4M0 in Chapter 8, "Event Log Analysis" on page 155
- Added 12.3.4, "NFS errors" on page 224, which includes information about debugging NFS errors. The content was provided originally in the Blue Gene/P System Operations Guide that is located in the Blue Gene Knowledge Base.
- Added 12.4.1, "Customizing and building the ramdisk" on page 228 to describe the steps required to build a customized ramdisk
- Included a complete list of the options that are available for the Navigator's database configuration in "Database configuration" on page 23
- Added Appendix E, "Setting up the cross-compile environment" on page 267 to describe the actions that a system administrator must take to enable this new feature
- Described how to build and install Python and PyDB2 in Appendix F, "Setting up the Event Log Analysis Environment" on page 269

Modified information

- Updated various sections in Chapter 2., "Blue Gene Navigator" on page 17 to reflect changes made in V1R4M0.
- Updated 12.3.3, "Performance tips for the NFS file server" on page 222 with additional material from the Blue Gene/P System Operations Guide that is located in the Blue Gene Knowledge Base.
- Changed the description of CIOD_RDWR_BUFFER_SIZE in Table 12-3 on page 218 to reflect the minimum size requirement of 8096 bytes.

February 2009, Third Edition

This revision reflects the addition, deletion, or modification of new and changed information described in the following sections.

New information

- V1R3M0 updates to the real-time server are in 3.6, “Real-time server” on page 94
- The improvements to RAS event reporting in V1R3M0 are outlined in Chapter 7, “RAS messages” on page 143
- Additional diagnostic test buckets are included in Chapter 9, “Diagnostics” on page 167
- Information about how to submit a Diagnostics run from a scheduler is included in 9.5, “Running diagnostics through a scheduler” on page 182
- A discussion about how to limit the impact of running diagnostics on the service node is included in 9.6, “Diagnostics performance considerations” on page 184
- Additional Service Action commands were added in 11.2.2, “Additional commands” on page 202
- Appendix C, “Service connection” on page 245 details the use of the service connection to the I/O nodes

Modified information

- Chapter 2, “Blue Gene Navigator” on page 17 contains the V1R3M0 enhancements to the Navigator.
- Additional commands were added in Chapter 5, “Midplane Management Control System” on page 115 to accommodate compute node Linux®.

September 2008, Second Edition

This revision reflects the addition, deletion, or modification of new and changed information described in the following sections.

New information

- RAS Messages (Chapter 7, “RAS messages” on page 143)
- High Throughput Computing (Chapter 6, “High Throughput Computing” on page 137)
- V1R2M0 Diagnostic updates (Chapter 9, “Diagnostics” on page 167)

Modified information

- Blue Gene Navigator updates (Chapter 2, “Blue Gene Navigator” on page 17)
- Steps to cycle power are changed (11.3, “Cycling power” on page 210)

1

The evolution of Blue Gene

This chapter highlights some of the significant changes made in the evolution process of Blue Gene/L to Blue Gene/P.

1.1 Hardware changes

Blue Gene/P keeps the familiar, slanted profile that was introduced with Blue Gene/L. However, the increased compute power requires increased airflow, resulting in a larger footprint. Each of the air plenums on Blue Gene/P are just over 10 inches wider than the previous model's plenums. Additionally, each Blue Gene/P rack is approximately four inches wider. There are two additional Bulk Power Modules mounted in the Bulk Power enclosure on the top of the rack. Rather than a circuit breaker style switch, there is an on/off toggle switch to turn on the rack.

1.1.1 Packaging

Figure 1-1 illustrates Blue Gene/L packaging.

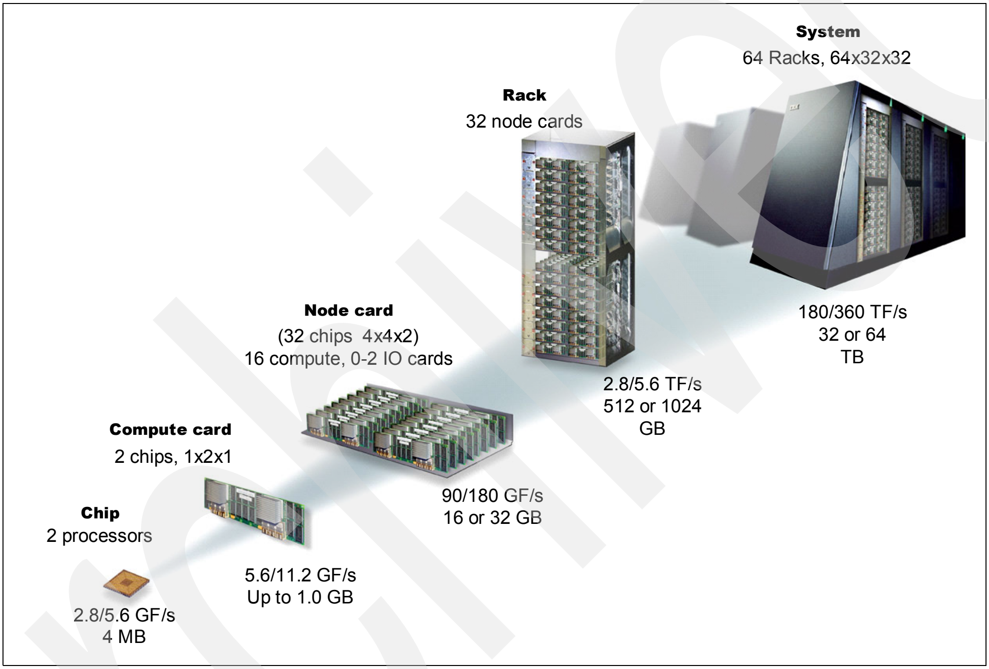


Figure 1-1 Blue Gene/L packaging

Figure 1-2 shows Blue Gene/P packaging.

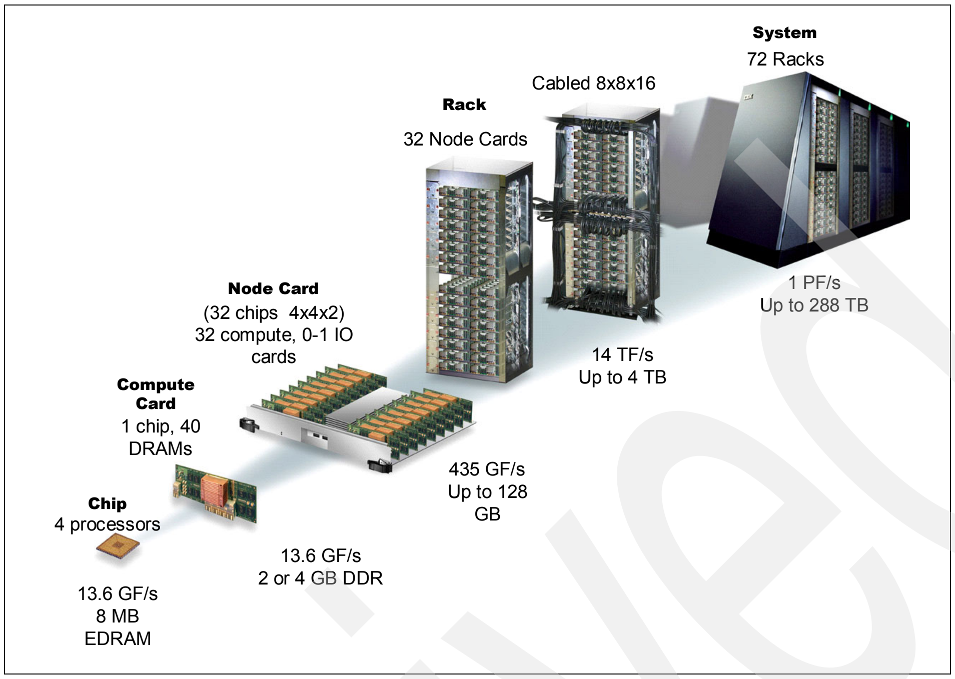


Figure 1-2 Blue Gene/P packaging

The changes start at the very bottom of the chain. Each chip is made of four processors rather than the two processors with Blue Gene/L. As you step up to the next level, notice that there is only one chip on each of the compute (processor) cards. Each of the compute cards can be equipped with either 2 or 4 GB of memory. This design is easy to maintain with less waste. Replacing a Blue Gene/L compute node because of a single failed processor meant discarding one usable chip (because they are packaged with two chips per card). The design of Blue Gene/P has only one chip per processor card, thus eliminating the disposal of a good chip when replacing a compute card.

With the Blue Gene/P, there are still 32 chips on each node card, but now the maximum number of I/O nodes per node card is two. So, there are only two Ethernet ports on the front of each node card. Similar to Blue Gene/L, there are two midplanes per rack. The lower midplane is considered to be the master. Each midplane drives one Service card, four Link cards, and 16 Node cards. Additionally there are 20 fan modules, which pull cooled air across the cards, that plug in to the rack's midplanes.

1.1.2 Power requirements

Each Blue Gene/P rack has its own independent supply, which either can be set for 480 V 3-phase input with a 100 A line cord that is connected to a dedicated circuit breaker or is wired for 200V/400V operation with a 175 A line card that is hard wired to an appropriately sized circuit breaker. On the rack, there are no circuit breakers, but there is an ON-OFF switch at the top of the exhaust plenum. 48 V DC power is supplied to the rack using nine 5 KW wide-ranging power supplies that cover both the US 208 V and 480 V 3-phase AC, as well as the 200 V power that is used elsewhere. The ninth power supply is redundant, because eight are enough to supply power to the rack. From that point, we use local, point of load DC-DC power supplies whenever the local power consumption is of order 50 W or more.

1.1.3 Cooling requirements

Each Blue Gene/P rack is cooled using 60 (20 sets of 3) 120 mm fans. These fans operate at a maximum speed of 6800 rpm pulling air across the midplane. Blue Gene/L has the same configuration of fans, but the peak speed is 6000 rpm. As with later versions of Blue Gene/L, the fans can be fine tuned to adjust the amount of airflow within the various areas inside the rack. Also, as we mentioned earlier, the racks are wider to provide better airflow across the cards in the rack.

1.1.4 Compute and I/O node

The compute nodes and I/O nodes in Blue Gene/L were two unique pieces of hardware. In Blue Gene/P, the two parts, although performing different functions, are actually interchangeable. Table 1-1 shows a comparison of the compute nodes.

Table 1-1 Comparison of Blue Gene/L and Blue Gene/P nodes

Feature	**Blue Gene/L**	**Blue Gene/P**
Cores per node	2	4
Core Clock Speed	700 MHz	850 MHz
Cache Coherency	Software managed	SMP
Private L1 cache	32 KB per core	32 KB per core
Private L2 cache	14 stream prefetching	14 stream prefetching
Shared L3 cache	4 MB	8 MB
Physical Memory per node	512 MB - 1 GB	2 - 4 GB
Main Memory Bandwidth	5.6 GBps	13.6 GBps
Peak Performance	5.6 GFlops per node	13.6 GFlops per node
Network Topologies	**Blue Gene/L**	**Blue Gene/P**
Torus Network		
Bandwidth	2.1 GBps	5.1 GBps
Hardware Latency (nearest neighbor)	200 ns (32B packet) and 1.6 µs (256B packet)	100 ns (32B packet) and 800 ns (256B packet)
Global Collective Network		
Bandwidth	700 MBps	1.7 GBps
Hardware Latency (Round trip worst case)	5.0 µs	3.0 µs
Full System (72 rack comparison)		
Peak Performance	410 TFlops	~1 PFlops
Power	1.7 MW	~2.3 MW

The compute nodes contain four PowerPC® 450 processors with 2 GB or 4 GB of RAM. It also runs a lightweight kernel to execute user-mode applications only. Typically, all four cores are used for computation either in Dual Node Mode, Virtual Node Mode, or Symmetric Multiprocessor Mode. Data is moved between the compute and I/O nodes over the Global Collective network.

The I/O nodes run an embedded Linux kernel with minimal packages that are required to support an NFS client, Ethernet network connections and to act as a gateway for the compute nodes in their respective rack to the external world. The I/O nodes present a subset of standard Linux operating interfaces to the user. The 10 Gigabit Ethernet interface of the I/O nodes is connected to the core Ethernet switch.

1.1.5 Networking updates

Blue Gene/P operates with the same five basic networks that were available on Blue Gene/L. The 3-dimensional Torus, collective, and global barrier networks provide the high performance intra-application communications. The Functional network allows the I/O nodes to send data to and receive data from the outside world, and the Service (or Control) network provides a link from the service node directly to the hardware.

To offload message handling overhead from the cores, injection and reception Direct Memory Access (DMA) is added to Blue Gene/P. In addition to reducing the load on the core, it is expected that network blockages will be prevented if the memory queues are not drained fast enough.

Blue Gene/L provided a 1 Gb Functional Ethernet. Blue Gene/P is upgraded to a 10 Gb Ethernet. The new Functional network is full duplexing and implements IPv4 checksums for both transmit and receive paths. On Blue Gene/P, the Functional network can consist of up to 32 links per midplane and can transmit up to 300 meters over 50 mm multimode fiber cable.

1.2 Software improvements

From a software perspective, the overall design of Blue Gene/P is relatively unchanged from Blue Gene/L. Many of the changes implemented were made to accommodate the new hardware and enhance existing features.

1.2.1 Location identification

Location strings are used to identify hardware in the Blue Gene rack. Blue Gene/L used multiple methods to locate hardware depending on whether it was being referred to by the baremetal code or the software code. For example, in Blue Gene/L you might see R001, or R00-M1, both referring to the same midplane. With Blue Gene/P the references to the various pieces of hardware are all consistent.

Figure 1-3 shows the convention used for Blue Gene/P. These locations can also be used as regular expressions in MMCS commands such as `write_con`.

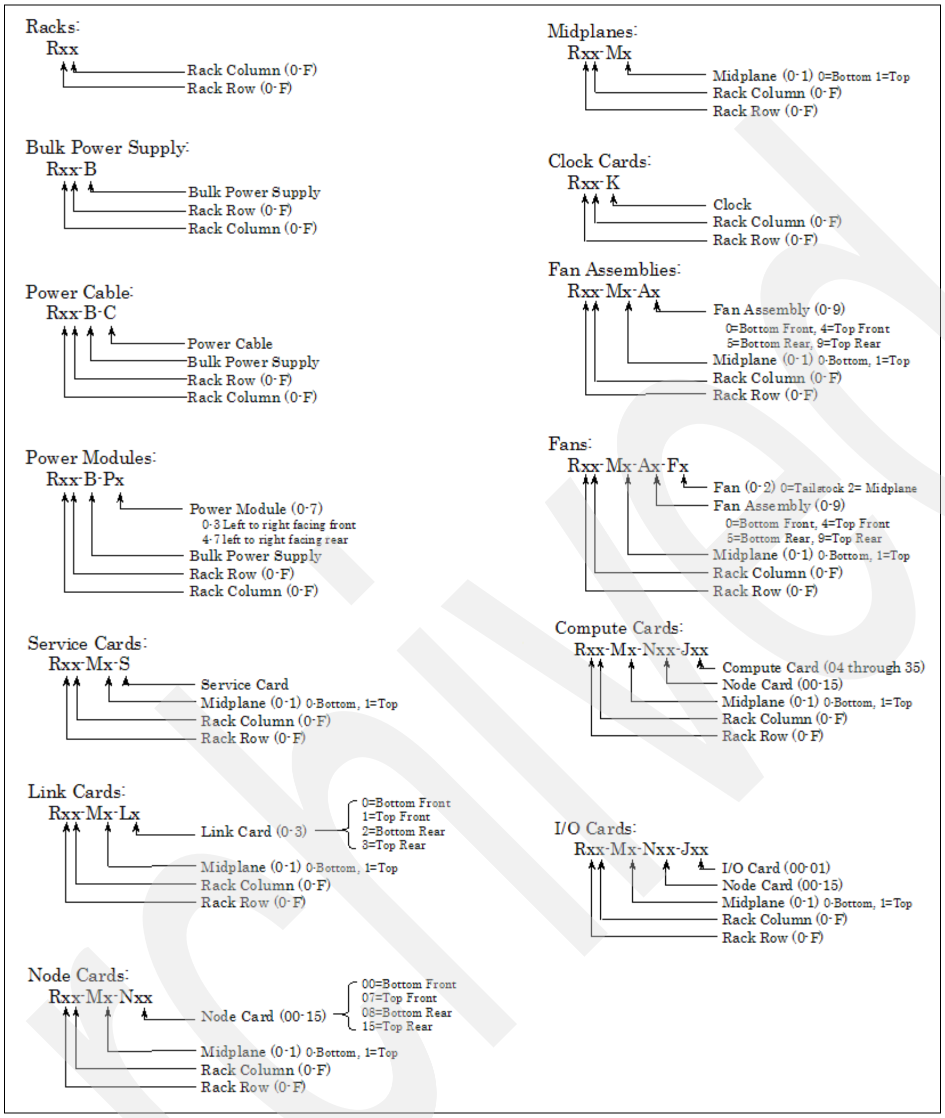
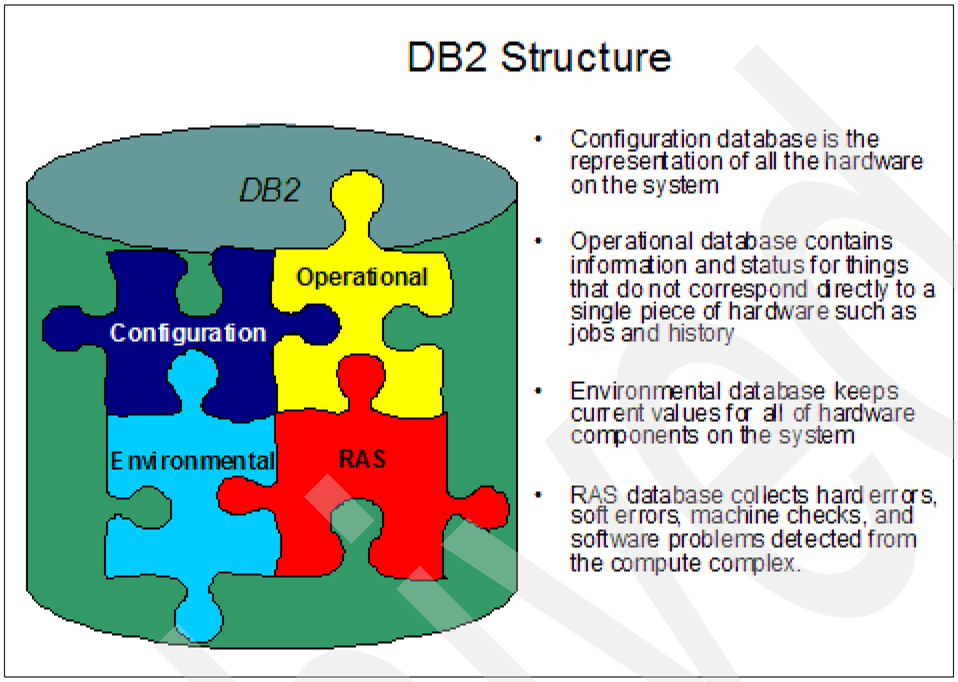


Figure 1-3 Blue Gene/P hardware naming convention

1.2.2 Database

The database has remained largely unchanged in Blue Gene/P. Figure 1-4 depicts the structure of the database on Blue Gene/P.



DB2 Structure

- Configuration database is the representation of all the hardware on the system
- Operational database contains information and status for things that do not correspond directly to a single piece of hardware such as jobs and history
- Environmental database keeps current values for all of hardware components on the system
- RAS database collects hard errors, soft errors, machine checks, and software problems detected from the compute complex.

Figure 1-4 Blue Gene/P database structure

One of the more notable improvements in the database schema was the change to the primary key in all of the hardware tables. The primary key used in Blue Gene/L was the serial number, which turned out to be somewhat cumbersome. Now, with Blue Gene/P, the location string is the primary key in the tables.

With Blue Gene/L, it was possible to show more than one piece of hardware for any given location on the system. For example, if a Link card was replaced, there would actually be two link cards associated with that location. Granted, only one could be active at a time, but if you were to do some action that required the hardware to be marked missing in the database, you would not be able to distinguish which entry in the database was for hardware that is currently installed in the system.

In Blue Gene/P, that will no longer be an issue. Parts that are replaced are still maintained in the database, but now they are stored in a history table (BGPREPLACEMENT_HISTORY table). So, from our example, the data for the currently installed Link card will be stored in the BPGLINKCARD table. The data for all of the Link cards that have been removed from the system will now reside in the BGPLINKCARD_HISTORY table.

There were some changes made to the database that stemmed from input by customers. One example of that is the BGPBLOCK table. The notion of processor sets (psets) was confusing to many. Now, rather than using the number of psets in the a block, we simply report the number of I/O nodes and compute nodes that are contained in a given block.

Database populate

After the Blue Gene/L system hardware installation was completed, a discovery script was run that polled all of the hardware and wrote information about that hardware back to the database. When a Blue Gene/P is installed, after the schema has been set up, a script called `dbPopulate.pl` is run. The dbpopulate script requires you to provide the number of racks in the system, and how the system is configured. For example, if you have an eight rack system, you need to specify if it is configured as a one by eight (1 x 8), two by four (2 x 4), or a four by two (4 x 2). The script then fills in the database with all the possible hardware that could be on the system.

InstallServiceAction

After the dbpopulate script runs, the *InstallServiceAction* program is run. In short, the InstallServiceAction program updates the database with information about the current state of the hardware (active, missing, or in error). Service actions might be necessary to replace hardware in error or missing.

VerifyCables

The Cable Discovery scripts on Blue Gene/L actually went out to the system and found each of the cables prior to updating the database. The Blue Gene/P dbpopulate scripts add the cables that should exist on the system to the database. The *VerifyCables* script's function is similar to the InstallServiceAction. It updates the database with information about the current state of the cables.

1.2.3 Reliability, Availability, and Serviceability infrastructure

Reliability, Availability, and Serviceability (RAS) has taken on a totally new format in Blue Gene/P. In Blue Gene/L, RAS was text based with no predefined format for RAS events. In Blue Gene/P, the RAS messages are very structured. There is a finite list of RAS messages that are used on Blue Gene/P, all of which have an identification (ID) number and a detailed message. The list of RAS events that have occurred or that might occur on your system are available for viewing through the Navigator. Additionally, there are 40 user-defined codes available that can be used to generate RAS messages that will be meaningful in the customer's environment.

1.2.4 Blocks

Blue Gene/P offers much more in the way of blocks, or *partitions* as they are also known. On Blue Gene/L, only small blocks with 32 or 128 nodes were supported due to the global interrupt wiring. With Blue Gene/P, you can configure any size small block (minimum of 16 compute nodes), as long as there is one I/O node contained within the block.

Small block allocation is optimized with the dynamic allocator. With the new version of the dynamic allocator the code looks at a Base Partition (BP) and if there is a node card missing the BP is marked unavailable for use with a large partition, but it will be made available for small block allocation. If an allocation attempt is made that requires resources equal to or less than the number of node cards that are available, the midplane is subdivided further. There is a minimal amount of fragmentation (orphaning) because of the optimization.

Because it is only possible to have a maximum of two I/O nodes per node card, the smallest I/O node to compute node ratio that is available is one to 16 (1:16). Blue Gene/L offered ratios as low as one to eight (1:8) because it was possible to have as many as four I/O nodes per node card.

1.2.5 Threading

The threading implementation on Blue Gene/P supports OpenMP. The XL OpenMP implementation provides a futex compatible syscall interface so the Native POSIX Thread Library (NPTL) pthreads implementation in glibc runs without modification. The compute node kernel defaults to allowing a total of four threads, one per CPU, and provides special thread function for I/O handling in MPI. Limited support for mmap is provided.

An environment variable, BG_APPTHREADDEPTH, can be used to override the default limit of one thread per CPU. Using this environment variable, the compute node kernel will allow up to 3 threads per CPU. These threads exhibit a fixed processor affinity, fixed/equal priority, and no implicit time-based preemption capability. Because of these limitations, only specialized applications can be run with more than one thread per processor configured.

1.2.6 Job modes

With Blue Gene/L, you had the choice of coprocessor mode or virtual node mode. Thus, your program could run on a single core in coprocessor mode, or you could split the memory and run your program on both cores in virtual node mode. With Blue Gene/P, you have three choices in this area. You can use *Symmetric Multi-Processing* (SMP) mode in which CPU 0 (MPI rank 0) runs the program's main process, as shown in Figure 1-5. The program can spawn up to three additional threads on the remaining processors in the default thread configuration.

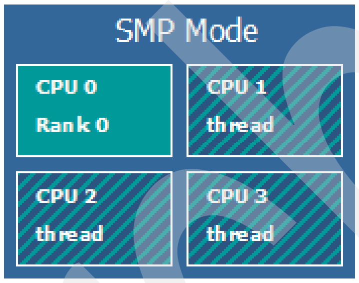


Figure 1-5 SMP mode

You can also use *Dual* mode. In Dual mode, CPUs 0 and 2 each run a main program process as shown in Figure 1-6. Each of those cores have an MPI rank and can spawn one additional thread in the default thread configuration. There is a fixed relationship between the cores. Thus, CPU 0 cannot start a thread on CPUs 1 and 3 and leave only the main process running on CPU 2. CPU 0 is only allowed send work to CPU 1, and likewise, CPU 2 can only use CPU 3 for additional threads.

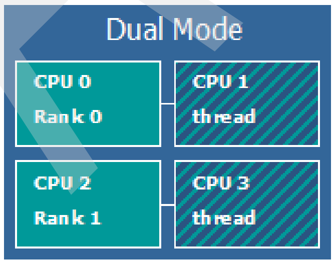


Figure 1-6 Dual processing mode

Finally, there is Virtual Node Mode (VNM), as shown in Figure 1-7. In this mode each of the cores in the processor have an MPI rank and run a program process. There is no additional threading capability in VNM when in the default thread configuration.

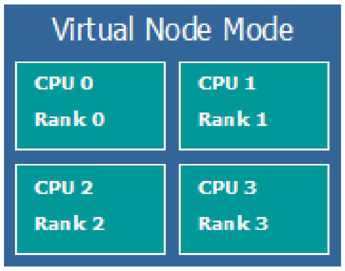


Figure 1-7 Virtual Node Mode

When submitting a High Performance Computing (HPC) job you can decide which mode your program runs in after the block is booted. In the HPC environment the mode is a job attribute (not a block attribute). The alternative, High Throughput Computing (HTC), requires that you specify a mode when you allocate an HTC partition.

Dual Mode and Virtual Node Mode require that you split the node's memory evenly between the main processes.

1.2.7 Control System

The Blue Gene/P Control System has undergone some streamlining that makes it more efficient. When referring to the Control System (illustrated in Figure 1-8), we are generally talking about the following components:

- Low Level Control System (LLCS)
  - Machine Controller (mc)
  - mcServer
- High Level Control System, which is also known as the Midplane Management Control System (MMCS)
- DB2® (RAS interfaces)
- Scheduler APIs
- Control and I/O Daemon (CIOD)

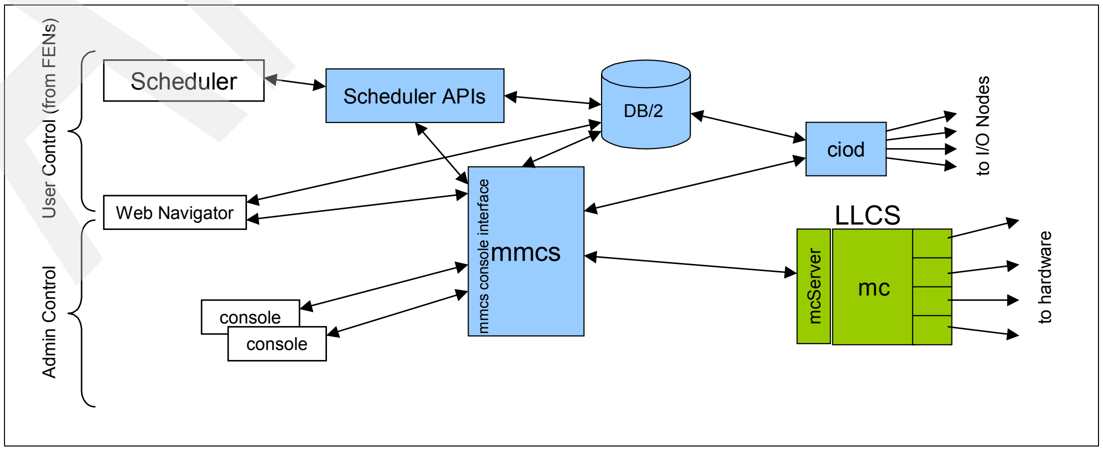


Figure 1-8 Blue Gene/P Control System

1.2.8 Proxy replaced

mcServer provides the low level access to the hardware for Blue Gene/P. This replaces the idoproxy functions that were used in Blue Gene/L. Much like the idoproxy, mcServer interfaces with the Machine Controller code to limit access to the hardware to one user at a time.

1.2.9 CIODB

Job submission requests from the various sources (schedulers, mpirun, and mmcs_db_console) update the job related database tables. On Blue Gene/L there is a separate process, CIODB, that runs on the service node and handles the job loading and starting. This code polls the job table, and starts jobs, monitors jobs, and updates the job table as the job goes through the states of being loaded, started, debugged, terminated, and so forth.

This code communicates with the I/O node daemon, CIOD, running on each I/O node, to control the job's progress and monitor status.

While this processing is very similar on Blue Gene/P, there is no longer a separate CIODB process. Because the CIODB code is so closely tied to mmcs_db_server, the two processes have been merged and run under the mmcs_db_server process.

1.2.10 Midplane Management Control System

Blue Gene/L Midplane Management Control System (MMCS) binaries were originally built in 32-bit mode but later mpirun (and libraries it depends on) were converted to 64-bit to provide support for the 64-bit schedulers that use the mpirun plug-ins. MMCS in Blue Gene/P has been built with 64 bit to avoid the problems encountered with the previous version.

On Blue Gene/L, you could direct certain commands such as **`write_con`** to a specific node by prefixing the command with the node's index as shown by the **`locate`** command, such as **`{0} write_con ps`**. On Blue Gene/P, regular expressions can also be used as prefixes. For example, **`{R00-M0-N04-J00} write_con ps`** is a valid command.

The **`connect`** command is new and can be used with a target specification. This command can be used to boot an individual node. Some of the parameters that were on the **`allocate_block`** command in Blue Gene/L have been moved to the **`connect`** command.

There is also a new option for MMCS, **`--ciod-persist`**, that allows job control connections to remain open between jobs for the life of the block, which eliminates the need to create a new TCP connection to CIOD every time a job starts.

The **`boot_block`** command on Blue Gene/P has different parameters from Blue Gene/L. You can specify the microloader, compute node and I/O node elf images on **`boot_block`** rather than issuing individual load commands.

Blue Gene/P also includes the ability to perform multiple connect-boot-disconnect sequences without reallocating the block. You can disconnect from the block's hardware resources without freeing the block and reconnect without reallocating the block. This feature allows you to boot the block several times without freeing it.

Reconnecting blocks and jobs

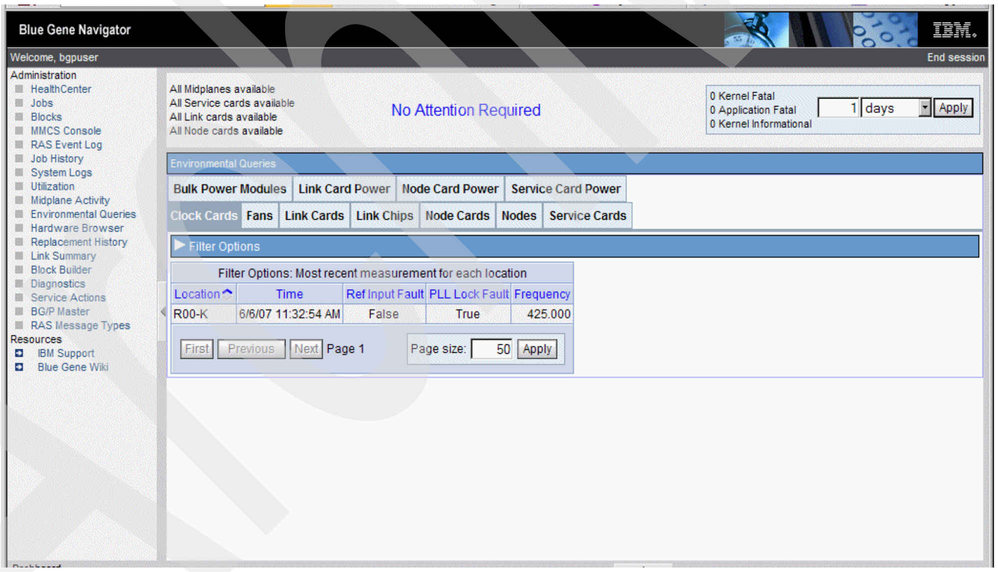
If CIODB ended for some reason while a job was running on a Blue Gene/L system, both the job and the block would stay active, but the ability to monitor and manage the job would be lost. Similarly, on Blue Gene/P if the mmcs_db_server server stops, the block stays booted, and the job keeps running on the nodes. With Blue Gene/P, you can allow reconnects when the server is restarted. This functionality is enabled simply by using the **`--reconnect`** option when starting the mmcs_db_server. When the server is restarted, the job can continue to write to stdout and stderr.

Note: Remember that any of the application's output that was sent to stderr or stdout while the server was down will be missing.

1.2.11 Hardware monitoring

Blue Gene/L had a separate process called *Hardware Monitor*. In Blue Gene/L, there were concerns about the amount of resources that were required to monitor the hardware. There were also several methods of starting the monitor functions on Blue Gene/L. On Blue Gene/P, the monitoring functions are integrated with the control system. The Environmental Monitor starts and ends with mmcs_db_server. The impact on the system as a whole is reduced, because there are only three connections to mcServer to collect the data at a time. The Environmental Monitor comes with the polling interval set to 5 minutes.

With the original Hardware Monitor, there were two different graphical user interfaces (GUIs) that could be used to view the data that was collected (Navigator and VNC). The results gathered by the monitor in Blue Gene/P can be accessed only through Blue Gene Navigator. Figure 1-9 shows the initial page that displays when Environmental Queries is selected in the Navigator.



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

No Attention Required

0 Kernel Fatal
0 Application Fatal
0 Kernel Informational

1 days Apply

Filter Options: Most recent measurement for each location

Location	Time	Ref Input Fault	PLL Lock Fault	Frequency
R00-K	6/6/07 11:32:54 AM	False	True	425.000

Figure 1-9 Environmental queries

In the Blue Gene/L version of the monitor, there was no limit on the length of time that collected data was stored. Monitoring and controlling the amount of data that was kept was entirely up to the administrators of the system. By default, Blue Gene/P purges the data every 3 months. The configuration can be altered to store more or less data as required by the local environment.

1.2.12 Client/server mpirun

The client/server mpirun on Blue Gene/P remains largely unchanged from Blue Gene/L. The most notable exception is the removal of the rsh/ssh mechanism for initiating the back-end process. Using the rsh or ssh protocols that are required that each user have a profile on the service node. In Blue Gene/P, a daemon process running on the service node handles connections from front-end mpirun processes and fork back-end (mpirun_be) mpirun processes.

Figure 1-10 shows how mpirun interacts with the remainder of the control system. After mpirun_be is forked, the sequence of events for booting partitions, starting jobs, and collecting stdout/stderr is very similar to Blue Gene/L mpirun.

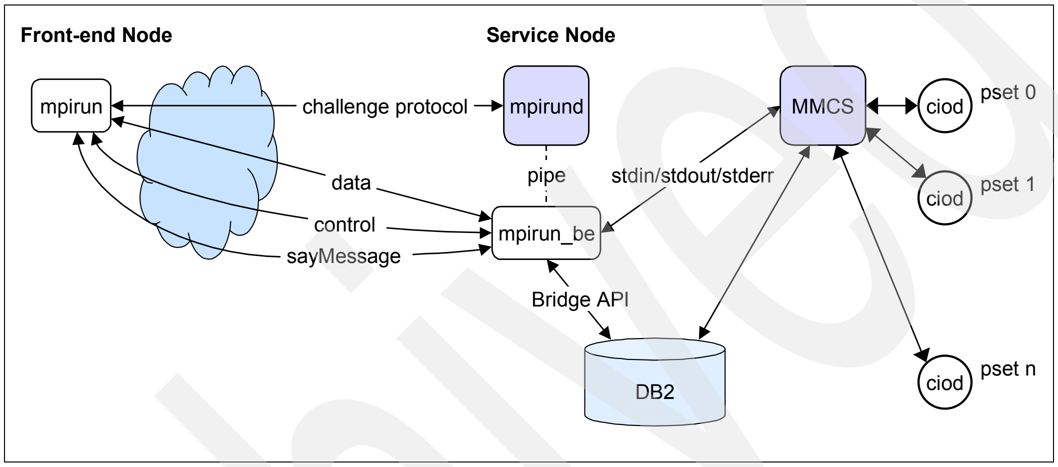


Figure 1-10 The mpirun flow

Another change in the Blue Gene/P version of mpirun is the support for Multiple Program Multiple Data (MPMD) style jobs. With MPMD a different executable, arguments, environment, and current working directory can be supplied for a single job on a processor set (pset) basis. For example, a user can execute four different executables on a partition that contains four psets. This function is handled by a new tool called *mpiexec*.

Note: The mpiexec tool should not be confused with the mpiexec style of submitting a Single Program Multiple Data (SPMD) parallel MPI job.

1.2.13 Bridge APIs

In Blue Gene/L, the Control System stored temporary XML files in the /tmp directory, which might cause issues occasionally if the /tmp file system became full. Blue Gene/P resolves this issue by no longer writing XML files to /tmp. Instead, they are passed in-memory. This change has improved overall Control System performance as well.

Users of the Bridge APIs (for example, mpirun and LoadLeveler®) should see significant performance improvements because of the implementation of database connection pooling and a reduction in the number of database connections that are required on some Bridge APIs.

1.2.14 Navigator updates

The original purpose of Blue Gene Navigator was to provide administrators with a GUI from which they could manage their system. Since that time, the Navigator has also evolved into a user tool that shows the health of the system, its utilization, and the availability of system resources. This section highlights some of the additional improvements made to the Navigator in Blue Gene/P.

Replacement History

One of the new items added to the Navigator is the Replacement History link. This link allows you to view all of the various hardware types that have been replaced and to filter your view based on specific times, location, serial number, or electronic chip ID (ECID).

Block Builder

The Block Builder feature introduced in the Blue Gene/L version of the Navigator allowed users to create blocks that consisted of one or more midplanes. In Blue Gene/L, the more complicated blocks such as sub-midplane (small) blocks needed to be defined using XML. Blue Gene/P has eliminated the need for the customers to do any XML. All of the necessary functionality is provided in the Navigator and the Bridge APIs, including small blocks and blocks doing pass through.

Service Actions

With Blue Gene/L, you could query the Service Actions table to view current or previous actions that have been performed on the system. Blue Gene/P allows administrators to actually initiate a Service Action from the Navigator.

BGPMaster

In the Blue Gene/P version of the Navigator, an interface to manage BGPMaster has been added. You can start, restart, or stop the mcserver, mmcs_server, mpirund, navigator_server, submit_server, and realtime_server from the GUI.

Status messages concerning each of the servers are displayed. For example, if a server has been stopped, the message shows what action is required to recover and how long the server has been down.

RAS Message Types

A link was added to the Navigator that allows you to search on the types of messages that have occurred on your system. The default query opens all message types that have occurred in the last day (24 hours), but you can filter to specific message IDs or time periods.

Plug-ins

The ability to change some of the information that is displayed by Blue Gene Navigator was added in Blue Gene/P. Now, you can write plug-ins to customize your interface. You can add to or replace the existing graphs at the bottom of the page. You can also add additional links to the navigation pane. By default, the Attention area at the top of the page alerts you when certain events occur, such as hardware failures or RAS events. You can customize this area to notify you of events that can have significance in your environment.

1.2.15 Parts replacement

Performing service actions on Blue Gene/P will not have the same impact that it did on Blue Gene/L. Originally, even changing a compute node required that administrators take an entire midplane out of service for a short time. In Blue Gene/P, the service actions are more finite. They only affect the specific hardware that is being serviced.

A good example of that is the same procedure that replaces a compute node. Now, this process only affects partitions (and jobs running on them) that include the specific node card that contains that compute node rather than the whole midplane. An even better example is the service action that is performed to replace a Link card. On a multiple rack system, this type of service would require that all the racks in the same row, and that all the racks in the column, be put into service mode. The Blue Gene/P Service Actions no longer need to turn off all of the neighboring link cards, resulting in a dramatic difference in the amount of time that it takes to prepare for and end the service action.

The Blue Gene Navigator has an interface that allows you to perform service actions and to also see which jobs and blocks will be impacted by the service action that you are going to perform.

1.2.16 Diagnostics

Diagnostics on Blue Gene/P are similar to the set of tests that were available on Blue Gene/L with the addition of test cases added to exercise new features such as DMA.

Still initiated from Blue Gene Navigator, you can track the progress and cancel a diagnostic run if necessary. Results are stored in the database and viewable from the Navigator. There are cross-reference links between RAS messages, failures, and hardware locations.

You can select between small, medium, or large sets of tests plus an additional test bucket added that is labelled *complete*. The complete option includes all the tests in the large suite, plus a single node Linpack and a torus connectivity test.

Diagnostics are more efficient in Blue Gene/P because pipelining has been introduced into the harness. The harness runs tests on the hardware while compiling results from previously run tests.

RAS and error reporting is improved for both diagnostics and normal system usage. For example, RAS events are now used to indicate failures as opposed to error messages being sprinkled throughout test output. This event-driven model provides many benefits:

- Reduced time spent parsing code
- Improved diagnostics runtime
- Decreased network utilization

The new RAS subsystem provides not only low-level error reporting, but decoding facilities in the control system to enhance error messages on the way back up to the user. Diagnostics also have a total view of all the RAS events posted during a test and has the ability to make decisions based on all the information presented.

Blue Gene Navigator

This chapter provides an overview of the Blue Gene Navigator. We discuss some of the features of the Navigator and give instructions about how to install and modify the interface.

2.1 Blue Gene Navigator overview

Blue Gene Navigator is designed to be a full-featured graphical user interface (GUI) that supports the administration of the Blue Gene/P core. You can also configure Navigator to be a user-oriented interface that is useful in providing information about usage, jobs, and system availability to those who might be submitting jobs to the Blue Gene/P core. Navigator is installed with the system software. The administrative view is the default set of pages that are enabled when the system is installed.

BGPMaster controls the Tomcat HTTP server on which Navigator runs. By default, the instance requires that Secure Sockets Layer (SSL) be configured and that the server listens on port 32072. Administrators access the page at:

`https://`*`hostname`*`:32072/BlueGeneNavigator`

The Navigator runs with the profile of the user that starts BGPMaster. In most environments, this profile is the *bgpadmin* user. The user that starts BGPMaster must have read access to the `/etc/shadow` files to allow the Navigator to perform authentications.

The supported browsers for Blue Gene Navigator are Firefox 2.0 and Internet Explorer® 7.0 or later.

2.1.1 Setup

The installation of the Blue Gene/P system covers the initial setup of the Navigator. However, if you need to reinstall the system, you need to complete a few steps prior to starting the Navigator.

Setup script

The DB2 libraries must be made available to Tomcat so that it can access the database, and the JAAS plug-in for interfacing to Linux PAM must be made available so that Tomcat can authenticate users. A script is provided to do the setup. Example 2-1 shows the syntax for running the script.

Example 2-1 Run the setup script

```
$ cd /bgsys/drivers/ppcfloor/navigator
$ ./setup-tomcat.sh
```

By default, the setup shown in Example 2-1 configures the Navigator to allow only authenticated users to access the Web interface. To enable anonymous access to user pages, you need to copy the `web-withenduser.xml` file into Navigator’s configuration, as shown in Example 2-2.

Example 2-2 Set up anonymous user pages

```
$ cd /bgsys/drivers/ppcfloor/navigator
$ cp web-withenduser.xml catalina_base/webapps/BlueGeneNavigator/WEB-INF/web.xml
```

Navigator uses the bluegene PAM stack to authenticate users, so it must be set up by creating a file named `/etc/pam.d/bluegene`, as shown in Example 2-3.

Example 2-3 The /etc/pam.d/bluegene file

```
#%PAM-1.0
auth     include       common-auth
auth     required      pam_nologin.so
account include       common-account
password include       common-password
session include       common-session
```

Note: The Navigator runs as the user that starts BGPMaster, and that same user profile is used to perform authentications. For example, if bgpadmin starts BGPMaster, and `/etc/pam.d/bluegene` is configured to use `pam_unix2.so`, then bgpadmin must be able to read the `/etc/shadow` file. If this is not the case, no one can sign on to Navigator.

SSL Configuration

By default, the Navigator uses the `/bgsys/local/etc/navigator.keystore` keystore. This keystore must be created when the system is configured using the **`keytool`** command as shown in Example 2-4. Replace *`myhostname.example.com`* with your service node's host name. The file should be owned by bgpadmin group.

Note: You can use a different password than *changeit*, but you will have to edit the `/bgsys/drivers/ppcfloor/navigator/catalina_base/conf/server.xml` file and add the following line to the Connector element with `port="32072"`:

```
keystorePass="newpass"
```

Because this file is part of the driver, you have to make this update every time a new driver is installed.

Example 2-4 The keytool command

```
$ keytool -genkey -alias tomcat -keyalg RSA -keystore
/bgsys/local/etc/navigator.keystore
Enter keystore password: changeit
What is your first and last name?
  [Unknown]: myhostname.example.com
What is the name of your organizational unit?
  [Unknown]:  STG
What is the name of your organization?
  [Unknown]:  IBM
What is the name of your City or Locality?
  [Unknown]:  Rochester
What is the name of your State or Province?
  [Unknown]:  MN
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=myhostname.example.com, OU=STG, O=IBM, L=Rochester, ST=MN, C=US correct?
(type "yes" or "no")
  [no]:  yes

Enter key password for <tomcat>
        (RETURN if same as keystore password):  changeit
```

```
$ chgrp bgpadmin /bgsys/local/etc/navigator.keystore

$ chmod 660 /bgsys/local/etc/navigator.keystore
```

The Navigator can be configured to accept non-SSL connections by editing the `/bgsys/drivers/ppcfloor/navigator/catalina_base/webapps/BlueGeneNavigator/WEB-INF/web.xml` file. To disable SSL all occurrences of the line `<transport-guarantee>CONFIDENTIAL</transport-guarantee>` need to be changed to `<transport-guarantee>NONE</transport-guarantee>`. The server must be restarted for the changes to take effect. Each time the Blue Gene software is upgraded the web.xml file must be edited, setting the transport-guarantee parameter to NONE.

The default non-SSL port is 32071. To access the Navigator non-securely use the URL `http://`*`hostname`*`:32071/BlueGeneNavigator`.

Block Visualizer

The Blue Gene Navigator administrative Web application has a new feature that allows visualization of blocks in the Block Details view. This feature is enabled by installing the JavaView applet to the Navigator Web application directory.

To install the JavaView applet, follow these steps:

1. Download the JavaView applet from the JavaView Web site. From a browser, navigate to:

   http://www.javaview.de

2. Click the **Download** link, and select the `javaview.zip` file to download.
3. Decompress `javaview.zip` in a temporary directory using the following command:

   ```
   bgpadmin@YourSys:/tmp> unzip javaview.zip
   ```

4. Copy the `jars/javaview.jar` file to the following directory:

   ```
   /bgsys/drivers/ppcfloor/navigator/catalina_base/webapps/BlueGeneNavigator/resou
   rces/jar
   ```

5. Make sure the file is readable by bgpadmin. Restart the Navigator if it is running when the file is copied into the directory. Use the following command:

   ```
   bgpadmin@YourSys>/bgsys/drivers/ppcfloor/bin/bgpmaster restart navigator_server
   ```

Authority

There are three roles defined in Navigator:

- End User
- Service
- Administrator

The Linux group to Navigator role mapping is defined in the local Navigator configuration file. The Administrator group has access to all pages. Table 2-1 describes the pages that Navigator allows users in the End User and Service roles.

Table 2-1 Group authorizations

Navigator link	Page	End User access	Service access
BG/P Master	about.jsp		X
	bgMaster.jsp		

Navigator link	Page	End User access	Service access
Blocks	blocks.jsp	X	X
	blockDetails.jsp	X	X
Block Builder	blockbuilder.jsp		
Diagnostics	diagnostics.jsp		X
	configDiags.jsp		X
	diagsBlock.jsp		X
	diagsLocation.jsp		X
	diagsLocTest.jsp		X
	diagsLog.jsp		X
	diagsRun.jsp		X
	diagsTestcase.jsp		X
Environmental Queries	environmentals.jsp		X
RAS Message Types	errCodes.jsp	X	X
Hardware Browser	hardwareBrowser.jsp		X
Replacement History	hardwareReplacement.jsp		X
Health Center	healthcenter.jsp	X	X
Jobs	jobs.jsp	X	X
	jobDetails.jsp	X	X
Job History	jobHistory.jsp	X	X
Link Summary	linkSummary.jsp		X
System Logs	systemLogs.jsp		X
	logFileView.jsp		X
Midplane Activity	midplaneActivity.jsp	X	X
MMCS Console	mmcs.jsp		
	mmcsblockoutput.jsp		
Service Actions	serviceActions.jsp		X
	prepareServiceAction.jsp		
RAS Event Log	ras.jsp	X	X
	rasDetails.jsp	X	X
Utilization	utilization.jsp		

2.1.2 Navigator options

This section describes options that are available to configure the Navigator to better suit your environment.

Startup options

Navigator is started from BGPMaster, which calls the `/bgsys/local/etc/startnavigator` script. BGPMaster passes several arguments to the `startnavigator` script. This script uses `baselogdir`, `useDatabase`, and `dbproperties`, as do the other startup scripts, to log the output. The script sets the `CATALINA_BASE` environment variable to `<`*`navigatorpath`*`>/catalina_base`. When `catalina.sh` is started, it uses this as the directory containing the tomcat instance files.

Several of the `startnavigator` options are turned into arguments and then passed to `catalina.sh`, which starts the Tomcat instance in which Navigator runs. When Navigator starts, it uses the arguments for its own configuration.

The `startnavigator` script uses the following options:

`--binpath`	Path to control system binaries, which defaults to current directory, usually `/bgsys/drivers/ppcfloor/bin`. Navigator calls <*`binpath`*>`/bgpmaster` when a user goes to the BG/P Master page.
`--configpath`	Path to configuration files, which defaults to `/bgsys/local/etc`. Navigator looks for the local Navigator configuration file (`navigator-config.xml`) here and `bgpmaster.init` (used on the BG/P Master page).
`--dbproperties`	`db.properties` file, which defaults to <*`configpath`*>`/db.properties` If this is set, Navigator reads this for the DB properties. Otherwise, it uses <*`configpath`*>`/db.properties`.
`--db2profile`	DB2 profile, which defaults to `db2profile`, but usually `~bgpsysdb/sqllib/db2profile`. The `startnavigator` script sources this file.
`--useDatabase`	Machine SN, which defaults to BGP. Navigator uses this to construct the default logs directory on the System Logs page.
`--baselogdir`	Path to logs, which defaults to `/bgsys/logs`.
`--navigatorpath`	Path to Navigator files, which defaults to <*`binpath`*>`/navigator`. Navigator looks for the floor navigator configuration file under this directory `navigator-config.xml`.

Options in the Navigator configuration file

The Navigator configuration files contain several options. The floor Navigator configuration file (usually `/bgsys/driver/ppcfloor/navigator/navigator-config.xml`) contains the default values. The default values can be overridden by putting the same option in your local navigator configuration file (`/bgsys/local/etc/navigator-config.xml`) with the new value.

You must create a local Navigator configuration file for authentication if nothing else. A sample local Navigator configuration file is shipped in `/bgsys/drivers/ppcfloor/navigator/sample-local-navigator-config.xml`. You need to copy this file to `/bgsys/local/etc` and edit it with the administrator, service, and user groups that are defined in your local configuration.

Administrator groups

Users in these Linux groups have access to all the sections in the Navigator. To have multiple groups, add an `<administrator-group>`*`groupname`*`</administrator-group>` for each group. Note that the value can be a Linux group name or gid (Example 2-5).

Example 2-5 Administrator configuration file entry

```
<administrator-group>bgpadmin</administrator-group>
```

Service groups

Users in these Linux groups have access to the service sections in the Navigator. To have multiple groups, add a `<service-group>`*`groupname`*`</service-group>` for each group. Note that the value can be a Linux group name or gid (Example 2-6).

Example 2-6 Service configuration file entry

```
<service-group>bgpservice</service-group>
```

User groups

Users in these Linux groups have access to only the end-user pages in the Navigator. These pages do not allow any updates to the Blue Gene system. To have multiple groups, add a `<user-group>`*`groupname`*`</user-group>` for each group. Note that the value can be a Linux group name or gid (Example 2-7).

Example 2-7 User group configuration file entry

```
<user-group>bgpuser</user-group>
```

Default chart timeout

This option is the timeout, in minutes, for the charts that are displayed in the Dashboard when the chart configuration does not set a timeout (Example 2-8).

Example 2-8 Chart timeout value

```
<default-chart-timeout>10</default-chart-timeout>
```

Plug-in library path

The plug-in library path provides the directory in which the Navigator looks to find plug-in libraries (jars), as shown in Example 2-9.

Example 2-9 The plugin-library-path entry

```
<plugin-library-path>/bgsys/local/lib</plugin-library-path>
```

Database configuration

Example 2-10 shows the default database configuration in Navigator's configuration file. For most environments these do not require any modification. The *min-idle-conns* and *initial-conns* are static connections to the database. The connections are maintained to reduce the connection startup overhead when a query is required.

Example 2-10 Database configuration options

```
<database>
   <driver-class>com.ibm.db2.jcc.DB2Driver</driver-class>
   <min-idle-conns>2</min-idle-conns>
   <initial-conns>2</initial-conns>
   <!--  Can also specify extra-params, extra parameters on the JDBC connect 
string. -->
   <!--  Can also specify system, the system name on the connect string. -->
</database>
```

The following sections list options for the database configuration.

The driver class option

The `driver-class` option is the database driver. If not specified, the default is `com.ibm.db2.jcc.DB2Driver`. Example 2-11 shows the `driver-class` entry in the configuration file.

Example 2-11 Driver class option

```
<driver-class>com.ibm.db2.jcc.DB2Driver</driver-class>
```

Extra connection parameters

You can add extra parameters to the connection string. Many options are available. Example 2-12 shows a parameter to enable tracing. The default is blank. For information about other options, see the DB2 Information Center at:

http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.apdv.java.doc/doc/rjvdsprp.htm

Example 2-12 Extra connection parameters

```
<extra-params>traceLevel=ALL</extra-params>
```

Minimum idle connections

The minimum idle connections parameter is the number of database connections that Navigator keeps open. The default is two. Use the syntax shown in Example 2-13 to change the parameter.

Example 2-13 Minimum idle connection parameter

```
<min-idle-conns>4</min-idle-conns>
```

Initial connections

The `initial-conns` parameter sets the number of database connections that Navigator opens when it starts. The default is two. Use the syntax shown in Example 2-14 to change this value.

Example 2-14 Initial connections parameter

```
<initial-conns>2</initial-conns>
```

System

The `system` parameter is the system name. If this parameter is not set, then a DB2 Type 2 JDBC™ connection is made, which uses the local database catalog. If this parameter is set, then a DB2 Type 4 JDBC connection is made. The default for this parameter is not set. To change the parameter, use the syntax as shown in Example 2-15.

Example 2-15 System parameter

```
<system>localhost</system>
```

System label

The `system-label` value is displayed in the *Welcome* line at the top of the Navigator and in the window title. If this value is not set, then nothing is displayed in these areas. Example 2-16 shows the entry that you can use to set the system label.

Example 2-16 System label parameter

```
<system-label>Your System Name</system-label>
```

Environmental options

The Environmental options contain two parameters that are related to highlighting the temperature readings on the Environmentals page, that is the minimum and maximum temperature values. Any temperature reading less than the `min-temp` value is blue, while any temperature reading more than the `max-temp` value are red, with shades of blue, green, and red for values between the minimum and maximum values (see Example 2-17).

Example 2-17 Environmental limits

```
<environmentals>
  <min-temp>25.0</min-temp>
  <max-temp>50.0</max-temp>
</environmentals>
```

BGPMaster options

These options provide the client executable and port used to call bgpmaster. If the client executable does not start with a slash (/), then the <*binpath*> is prepended. A value of `default` for the port means that the default port for BGPMaster is used. See Example 2-18.

Example 2-18 BGPMaster options

```
<bg-master>
    <client-exe>bgpmaster</client-exe>
    <port>default</port>
  </bg-master>
```

MMCS console options

The Navigator's MMCS console interface provides nearly the same functionality that is available when you access the console from a shell session (see Example 2-19). The Navigator provides a few features that are either not available or that appear differently than the command-line version. For a more complete listing of functions, see 2.3.4, "Midplane Management Control System" on page 47.

The configurable options are:

Maximum Reply Size	MMCS messages larger than this value is truncated.
Maximum History Size	The maximum size of all the MMCS messages that display on the MMCS page. Replies are removed from the back of the history after this size is reached.
Redirect Applet Log Level	The log level for the Redirect Applet affects the level of logging in the Java™ console on the user's browser. The possible values are the typical severity levels that are found in `java.logging`.
mmcs_db_server options	The host name and the port that Navigator uses to connect to the mmcs_db_server.
Redirect DB Server options	The host name and the port that the Navigator uses to connect to the redirect server.

Example 2-19 MMCS console options

```
<mmcs-console>
    <max-reply-size>8029</max-reply-size>
    <max-history-size>2097152</max-history-size>
    <redirect-applet-log-level>INFO</redirect-applet-log-level>
    <mmcs-db-server>
      <hostname>127.0.0.1</hostname>
      <port>32031</port>
    </mmcs-db-server>
    <redirect-server>
      <hostname>127.0.0.1</hostname>
      <port>32032</port>
    </redirect-server>
  </mmcs-console>
```

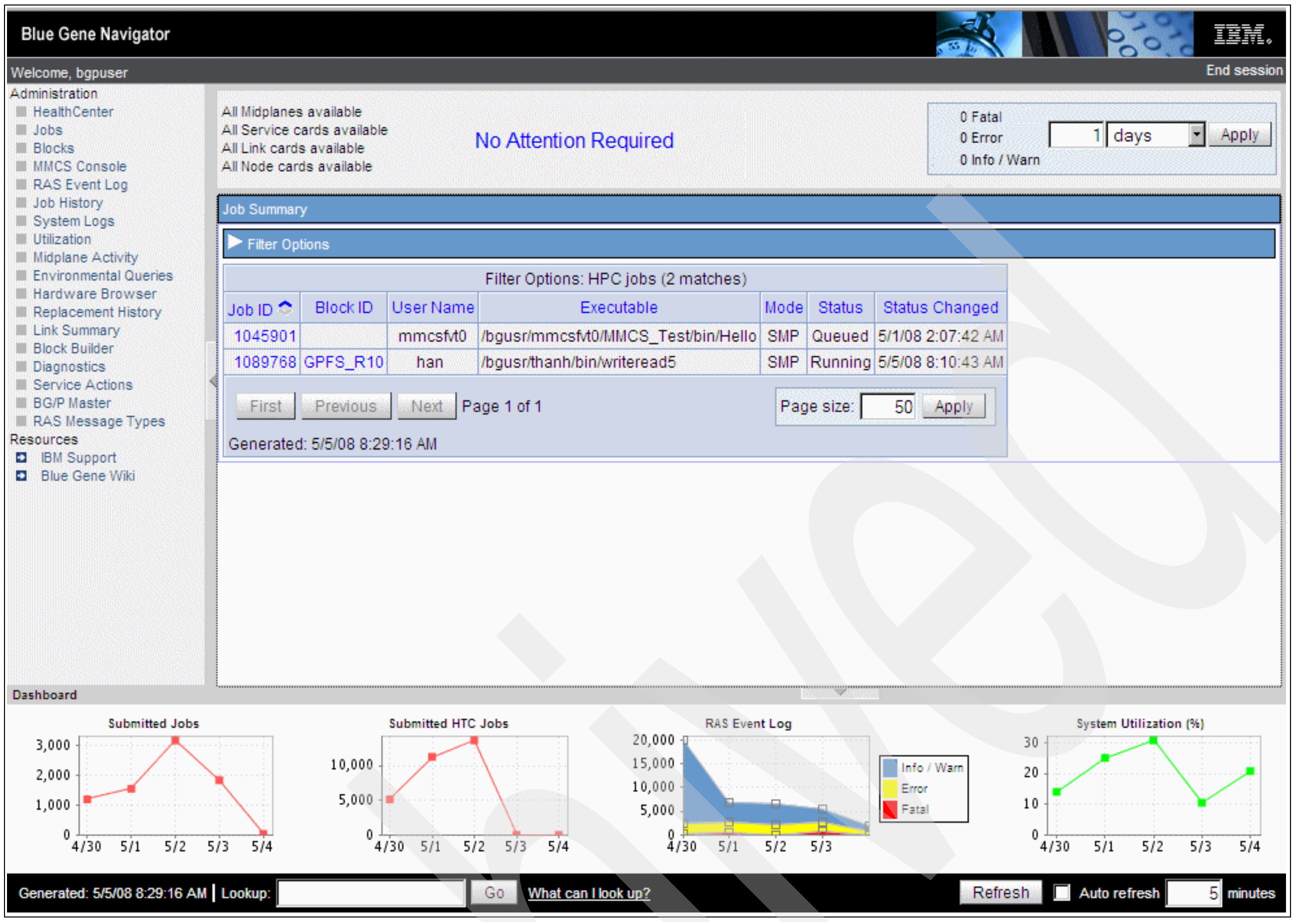
2.2 Common functions

Many features in the Blue Gene Navigator remain common throughout the entire interface. Although most of the features are intuitive, in this section we highlight these features and provide comments on their use.

Start by accessing the administrative version of Navigator to open the Jobs page shown in Figure 2-1 at the following Web address:

```
https://ServiceNodeName:32072/BlueGeneNavigator
```

This page is the default home page.



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

No Attention Required

0 Fatal
0 Error
0 Info / Warn

1 days Apply

Job Summary

Filter Options

Filter Options: HPC jobs (2 matches)

Job ID	Block ID	User Name	Executable	Mode	Status	Status Changed
1045901		mmcsfvt0	/bgusr/mmcsfvt0/MMCS_Test/bin/Hello	SMP	Queued	5/1/08 2:07:42 AM
1089768	GPFS_R10	han	/bgusr/thanh/bin/writeread5	SMP	Running	5/5/08 8:10:43 AM

First Previous Next Page 1 of 1 Page size: 50 Apply

Generated: 5/5/08 8:29:16 AM

Generated: 5/5/08 8:29:16 AM | Lookup: 5 minutes

Figure 2-1 Jobs page

Note: Figure 2-1 is shown with the HTC jobs graph. By default the HTC jobs graph is not configured on the system. You can find instructions about adding the chart in “High Throughput Computing Jobs graph” on page 37

The top, left of the page shows the user who is currently logged in, and on the right a link displays to end the session (as shown in Figure 2-2).

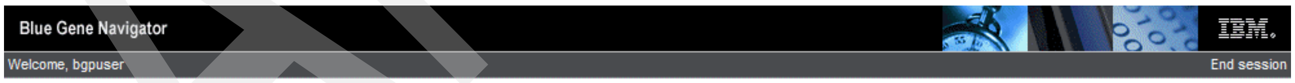
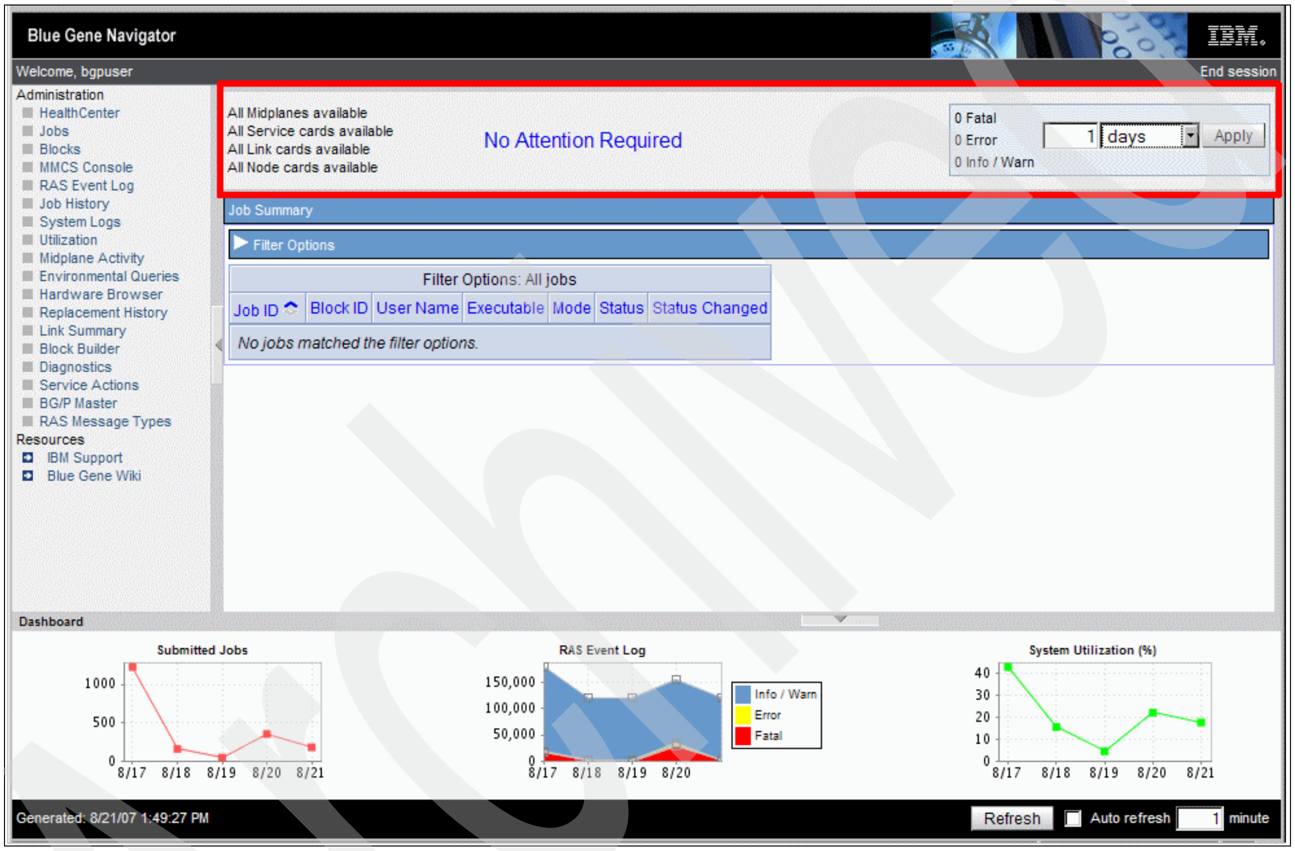


Figure 2-2 Top bar

Just below the title bar that shows the current user is the *Attention area* (Figure 2-3). On the left side of this area, the current status of the hardware in the Blue Gene/P core displays. If there are any issues, the number and type of hardware that is not available is reported in this section.

On the right side of this area, you can query the database for Kernel and Application errors based on time. Supply a number in the open selection box (default is 1 day) and then use the drop-down menu and select either seconds, minutes, hours, or days. Click **Apply** to run the query, and the results are returned and posted within the shaded box. Notice that as long as there are no errors reported, the *No Attention Required* message displays in the center of this section.



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

No Attention Required

0 Fatal
0 Error
0 Info / Warn

1 days

Filter Options: All jobs

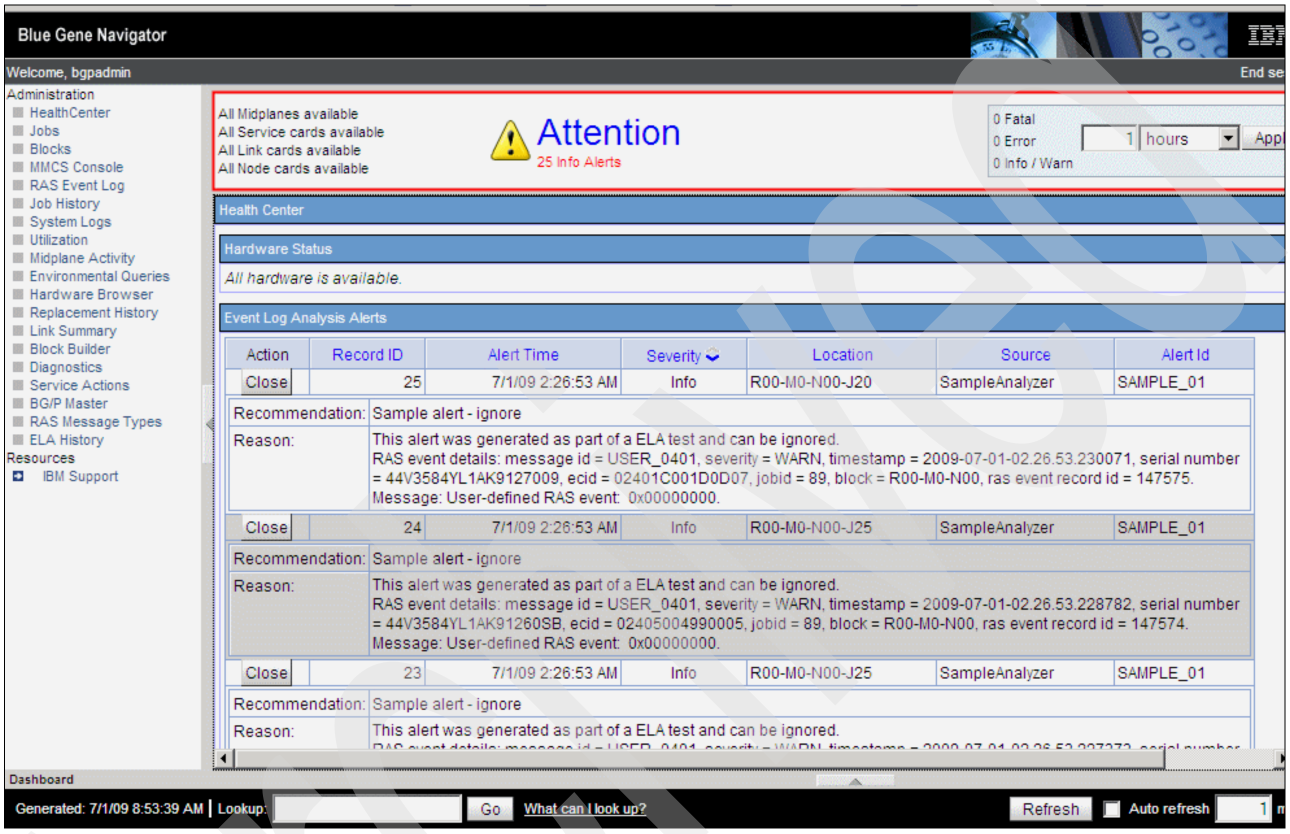
Job ID	Block ID	User Name	Executable	Mode	Status	Status Changed

No jobs matched the filter options.

Generated: 8/21/07 1:49:27 PM

Figure 2-3 Attention area

If there was a problem, the message in the center of the area changes to that shown in Figure 2-4. There is a default set of events that trigger the attention message that displays. You can customize the attention area of the Navigator to alert you based on events of your choice. (See 2.4, “Customizing Blue Gene Navigator” for information about how to do this). Clicking **Attention** takes you to the Health Center page so that you can locate the source of the problem. Clicking the Alerts text takes you directly to the Event Log Analysis Alerts section of the Health Center. (You can find more information about the Event Log Analysis tool in Chapter 8., “Event Log Analysis” on page 155).


Blue Gene Navigator

Welcome, bgpadmin

Administration
- HealthCenter
- Jobs
- Blocks
- MMCS Console
- RAS Event Log
- Job History
- System Logs
- Utilization
- Midplane Activity
- Environmental Queries
- Hardware Browser
- Replacement History
- Link Summary
- Block Builder
- Diagnostics
- Service Actions
- BG/P Master
- RAS Message Types
- ELA History

Resources
- IBM Support

All Midplanes available
All Service cards available
All Link cards available
All Node cards available

Attention
25 Info Alerts

0 Fatal
0 Error
0 Info / Warn

1 hours

Health Center

Hardware Status

All hardware is available.

Event Log Analysis Alerts

Action	Record ID	Alert Time	Severity	Location	Source	Alert Id
Close	25	7/1/09 2:26:53 AM	Info	R00-M0-N00-J20	SampleAnalyzer	SAMPLE_01

Recommendation: Sample alert - ignore
Reason: This alert was generated as part of a ELA test and can be ignored.
RAS event details: message id = USER_0401, severity = WARN, timestamp = 2009-07-01-02.26.53.230071, serial number = 44V3584YL1AK9127009, ecid = 02401C001D0D07, jobid = 89, block = R00-M0-N00, ras event record id = 147575.
Message: User-defined RAS event: 0x00000000.

Action	Record ID	Alert Time	Severity	Location	Source	Alert Id
Close	24	7/1/09 2:26:53 AM	Info	R00-M0-N00-J25	SampleAnalyzer	SAMPLE_01

Recommendation: Sample alert - ignore
Reason: This alert was generated as part of a ELA test and can be ignored.
RAS event details: message id = USER_0401, severity = WARN, timestamp = 2009-07-01-02.26.53.228782, serial number = 44V3584YL1AK91260SB, ecid = 02405004990005, jobid = 89, block = R00-M0-N00, ras event record id = 147574.
Message: User-defined RAS event: 0x00000000.

Action	Record ID	Alert Time	Severity	Location	Source	Alert Id
Close	23	7/1/09 2:26:53 AM	Info	R00-M0-N00-J25	SampleAnalyzer	SAMPLE_01

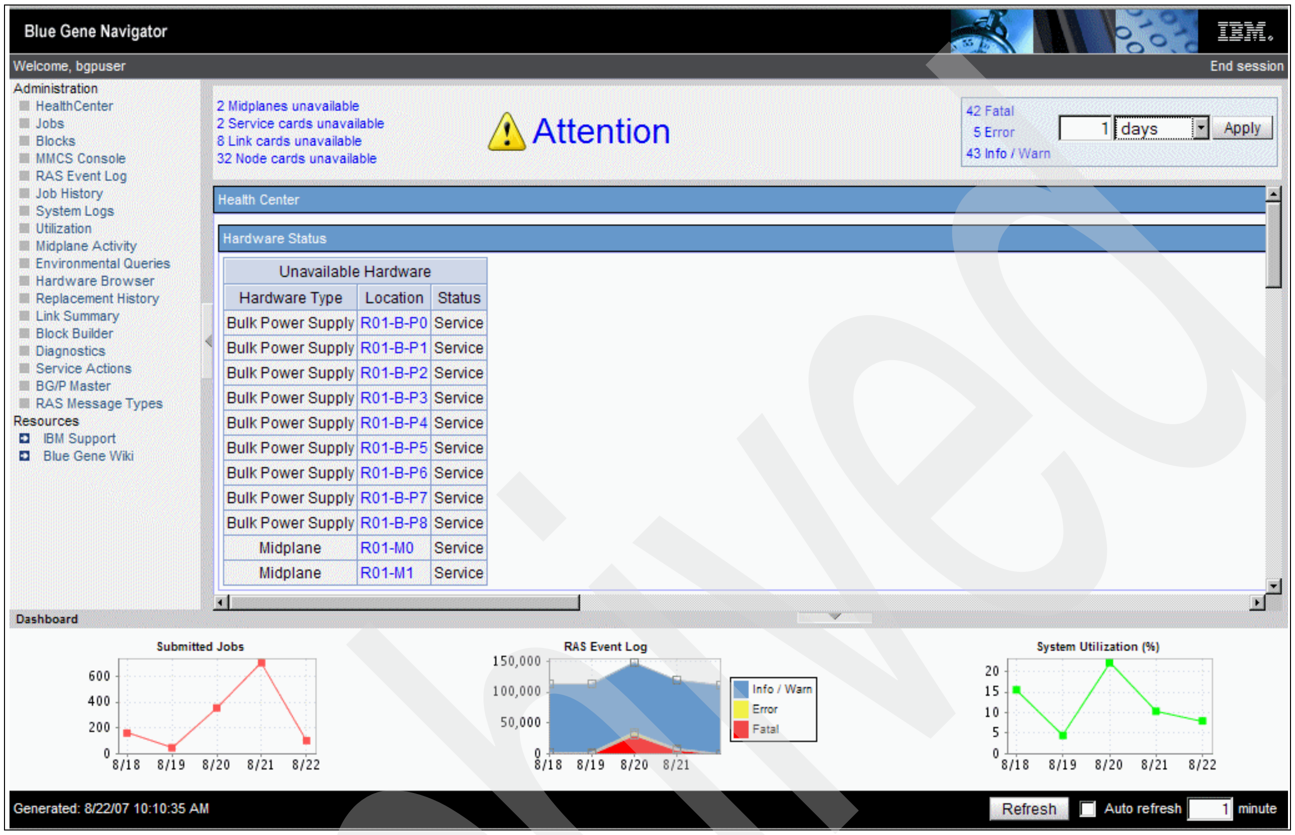
Recommendation: Sample alert - ignore
Reason: This alert was generated as part of a ELA test and can be ignored.

Dashboard

Generated: 7/1/09 8:53:39 AM | Lookup: Go What can I look up? Refresh Auto refresh 1

Figure 2-4 Attention required message

When you are in the Health Center, in the event that hardware is unavailable, you see a summary listed in the Attention area as shown in Figure 2-5. Clicking any of the affected hardware takes you to the Health Center page. From there, you can identify the individual pieces of hardware that are not available.



2 Midplanes unavailable
2 Service cards unavailable
8 Link cards unavailable
32 Node cards unavailable

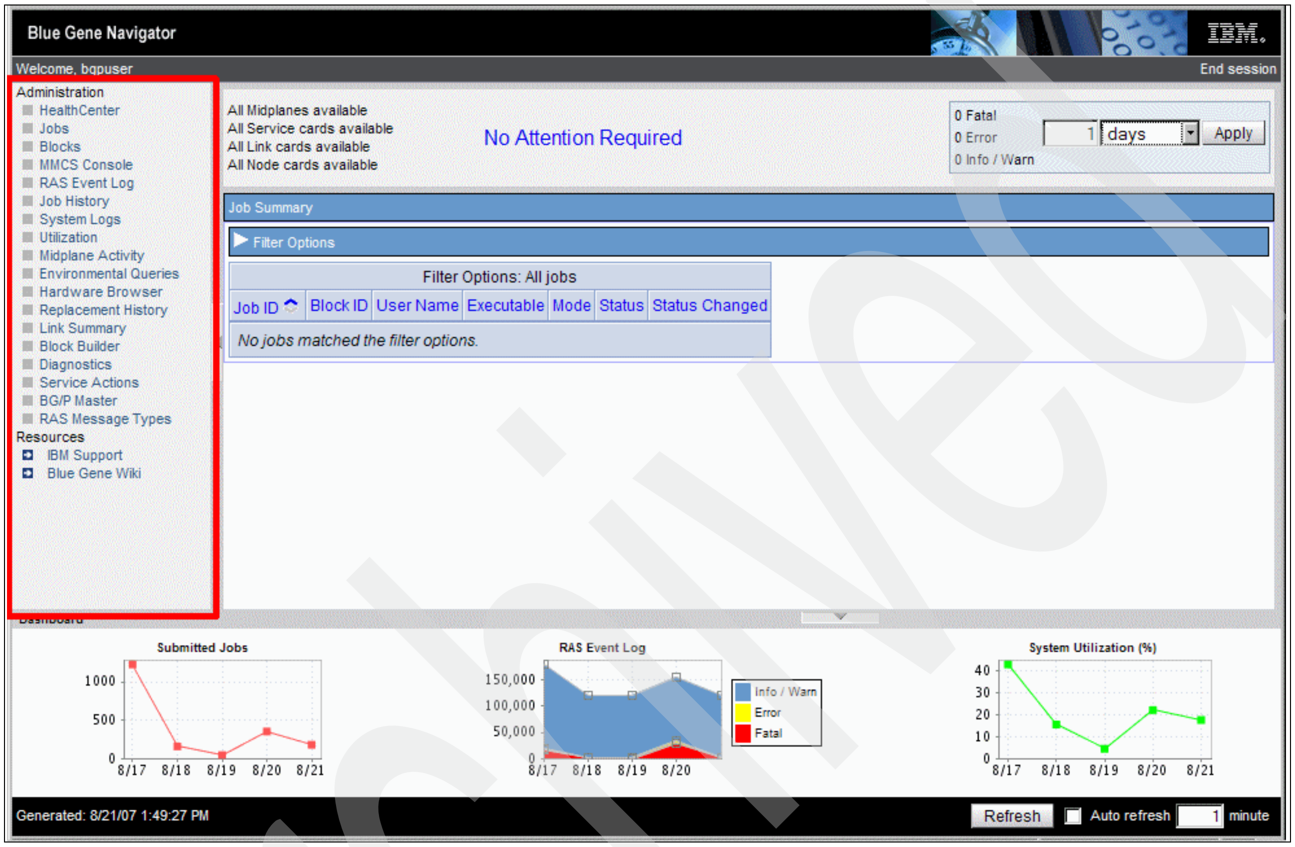
Unavailable Hardware

Hardware Type	Location	Status
Bulk Power Supply	R01-B-P0	Service
Bulk Power Supply	R01-B-P1	Service
Bulk Power Supply	R01-B-P2	Service
Bulk Power Supply	R01-B-P3	Service
Bulk Power Supply	R01-B-P4	Service
Bulk Power Supply	R01-B-P5	Service
Bulk Power Supply	R01-B-P6	Service
Bulk Power Supply	R01-B-P7	Service
Bulk Power Supply	R01-B-P8	Service
Midplane	R01-M0	Service
Midplane	R01-M1	Service

Generated: 8/22/07 10:10:35 AM

Figure 2-5 Unavailable hardware

On the extreme left of the panel is the *navigation pane* shown in Figure 2-6. Clicking any of the links under the Administration heading takes you to another page that is designed to administer or query that specific subject. You can find details about each of the links in 2.3, “Using the Navigator” on page 39. You can also customize the lower section of the navigation pane. You can add links that might be useful in your environment. By default, only the IBM Support link is provided.



Blue Gene Navigator

Welcome, bgpuser

End session

Administration
- HealthCenter
- Jobs
- Blocks
- MMCS Console
- RAS Event Log
- Job History
- System Logs
- Utilization
- Midplane Activity
- Environmental Queries
- Hardware Browser
- Replacement History
- Link Summary
- Block Builder
- Diagnostics
- Service Actions
- BG/P Master
- RAS Message Types

Resources
- IBM Support
- Blue Gene Wiki

All Midplanes available
All Service cards available
All Link cards available
All Node cards available

No Attention Required

0 Fatal
0 Error
0 Info / Warn

1 days Apply

Job Summary

Filter Options

Filter Options: All jobs

Job ID	Block ID	User Name	Executable	Mode	Status	Status Changed
No jobs matched the filter options.						

Dashboard

Generated: 8/21/07 1:49:27 PM

Refresh Auto refresh 1 minute

Figure 2-6 Navigation pane

After clicking a link in the navigation pane the content is loaded in the center frame. For consistency, we refer to this as the *content area* (Figure 2-7). There are several features in the content area that are available in most views.

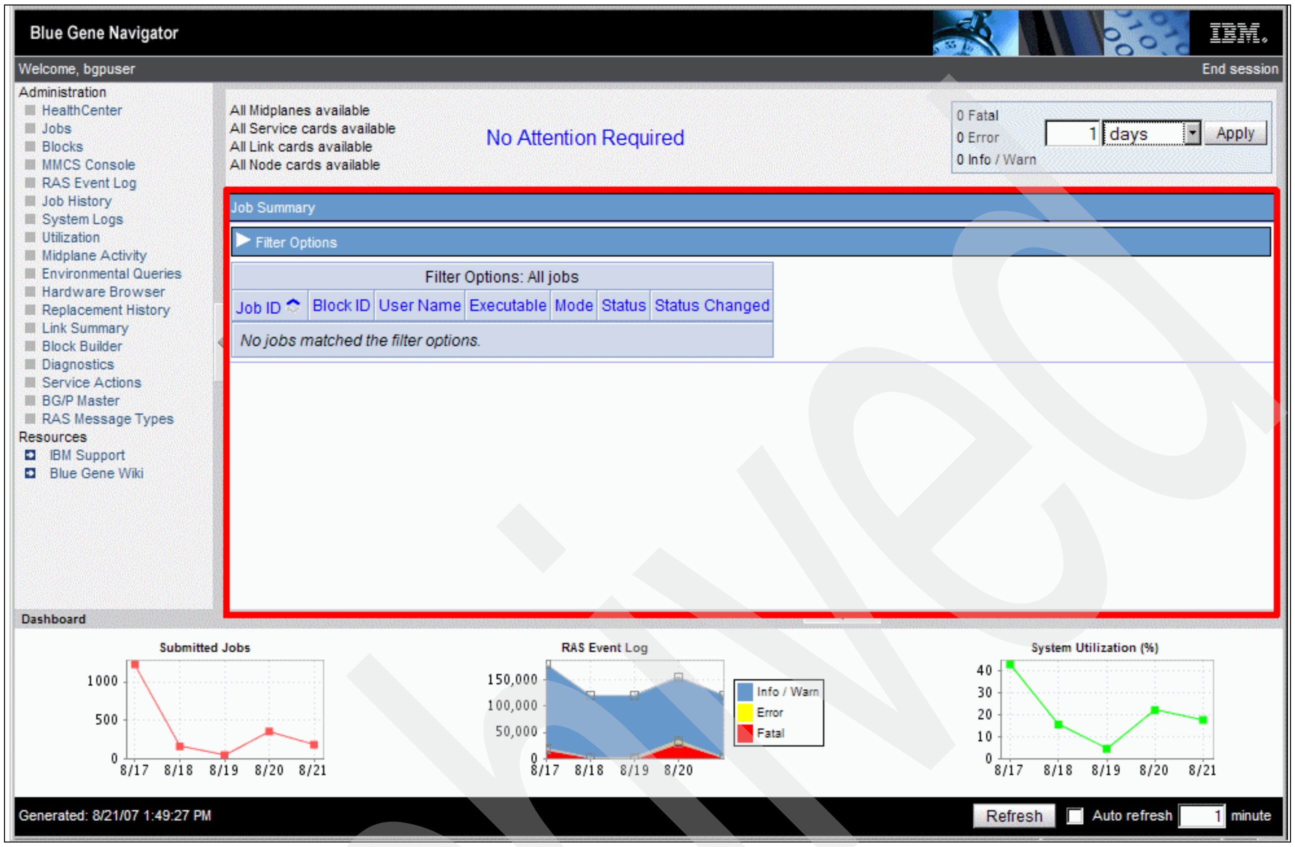
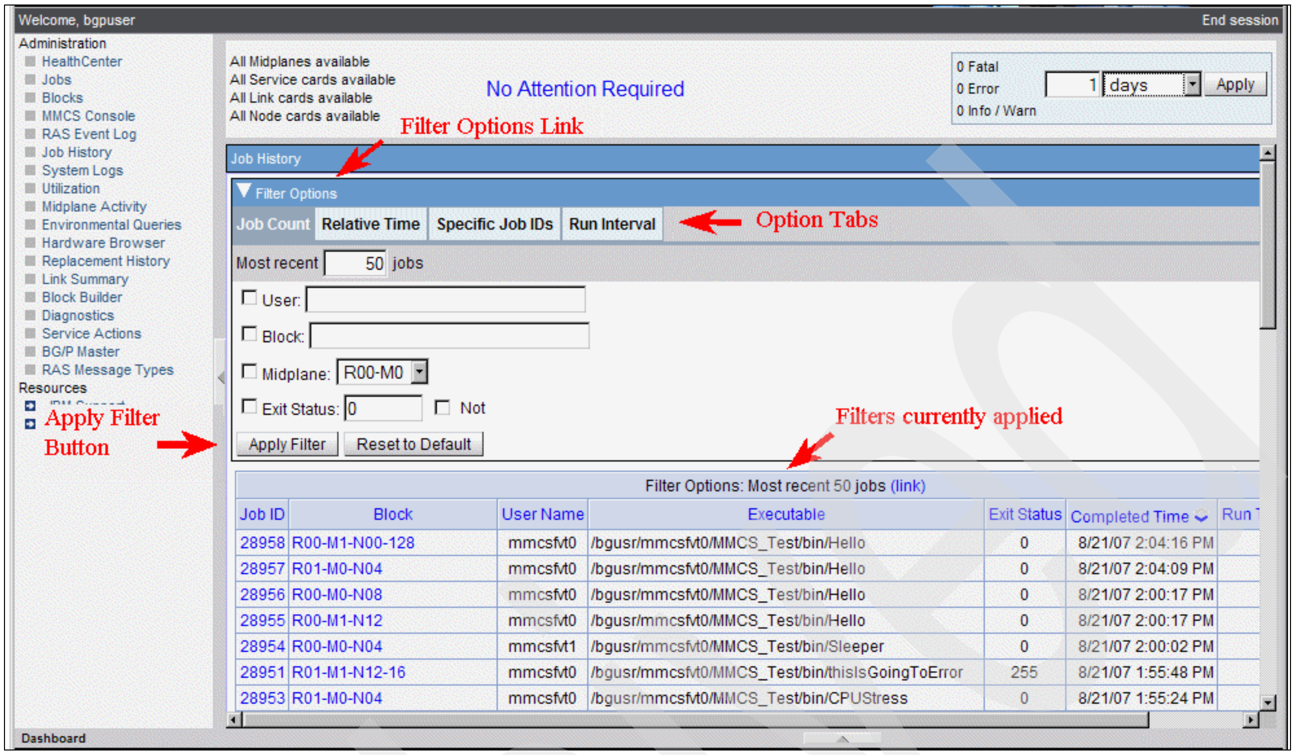


Figure 2-7 Content area

Almost every selection that you can make in the navigation page opens a page that has a Filter Options link. Clicking the white triangle (sometimes referred to as a *twisty*, shown in Figure 2-8) opens a display that gives you the available filter options for that panel.

Figure 2-8 Twisty icon

Figure 2-9 shows the Filter Options that are available for the Job History page.



Filter Options: Most recent 50 jobs (link)

Job ID	Block	User Name	Executable	Exit Status	Completed Time	Run
28958	R00-M1-N00-128	mmcsfvt0	/bgusr/mmcsfvt0/MMCS_Test/bin/Hello	0	8/21/07 2:04:16 PM	
28957	R01-M0-N04	mmcsfvt0	/bgusr/mmcsfvt0/MMCS_Test/bin/Hello	0	8/21/07 2:04:09 PM	
28956	R00-M0-N08	mmcsfvt0	/bgusr/mmcsfvt0/MMCS_Test/bin/Hello	0	8/21/07 2:00:17 PM	
28955	R00-M1-N12	mmcsfvt0	/bgusr/mmcsfvt0/MMCS_Test/bin/Hello	0	8/21/07 2:00:17 PM	
28954	R00-M0-N04	mmcsfvt1	/bgusr/mmcsfvt0/MMCS_Test/bin/Sleeper	0	8/21/07 2:00:02 PM	
28951	R01-M1-N12-16	mmcsfvt0	/bgusr/mmcsfvt0/MMCS_Test/bin/thisIsGoingToError	255	8/21/07 1:55:48 PM	
28953	R01-M0-N04	mmcsfvt0	/bgusr/mmcsfvt0/MMCS_Test/bin/CPUStress	0	8/21/07 1:55:24 PM	

Figure 2-9 Filter options

Notice on this page, just below the Filter Options title bar, that the following option tabs are available:

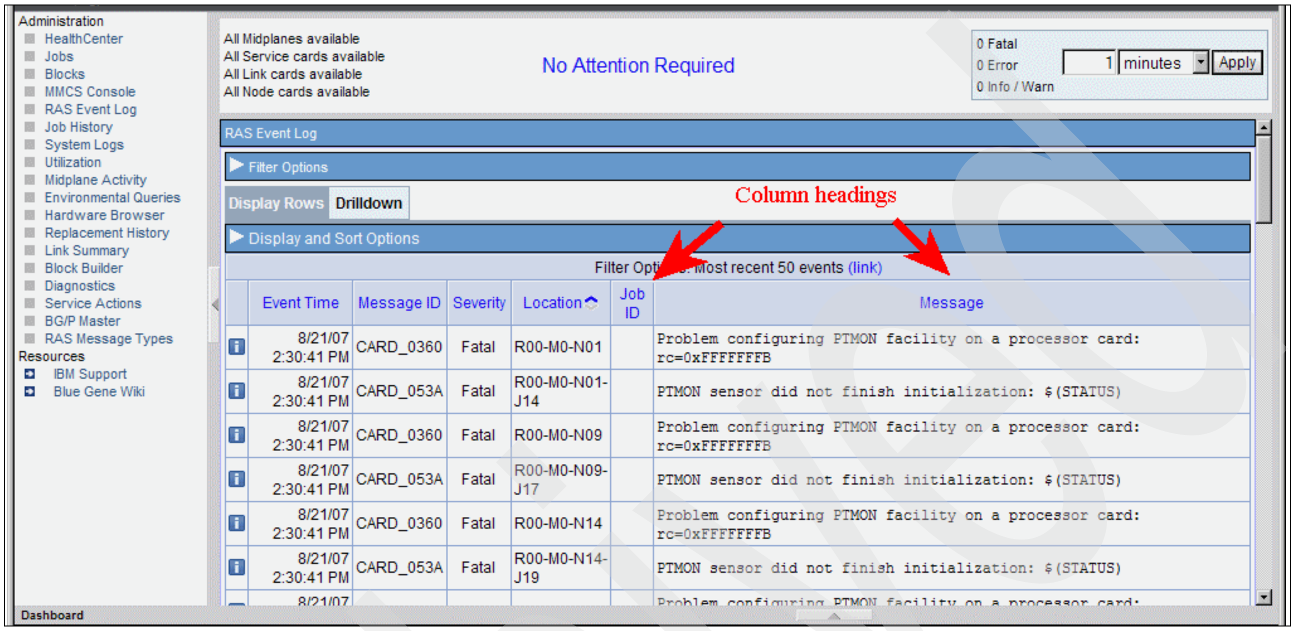
- Job Count
- Relative Time
- Specific Job IDs
- Run Interval

Clicking any one of these tabs gives a new set of options that you can use to filter the data that is returned. The number of tabs, and the options that are available on the tabs, are different in the various environments.

After you have narrowed the search by supplying the filters, select **Apply Filter** located at the bottom of the options to submit a query using the selected filters. When a filter is applied on a given page, it is stored in a session cookie. Every time you come back to that page, the same filter is applied unless you click **Reset to Default** or choose new filters.

Just below the Filter Options section of the page, there is a title bar that contains the current filters. After the list of options the hotspot (link) allows you to save the selected filter options in your browser's bookmarks. To do this, right-click the hotspot and select the option to bookmark the page. Selecting the saved bookmark reruns the query and returns the most current data that matches the filters.

Blue text in the Navigator indicates that it is a link to more information or provides a separate function. For example, clicking the column headings when they are blue sorts the data in the content portion of the page. In Figure 2-10, we sorted by clicking the *Location* column head. Now, the up arrow (^) is blue, indicating that the data is sorted in ascending order by location. Clicking the Location heading a second time sorts the data in descending order.



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

No Attention Required

0 Fatal
0 Error
0 Info / Warn
1 minutes

Filter Options: Most recent 50 events (link)

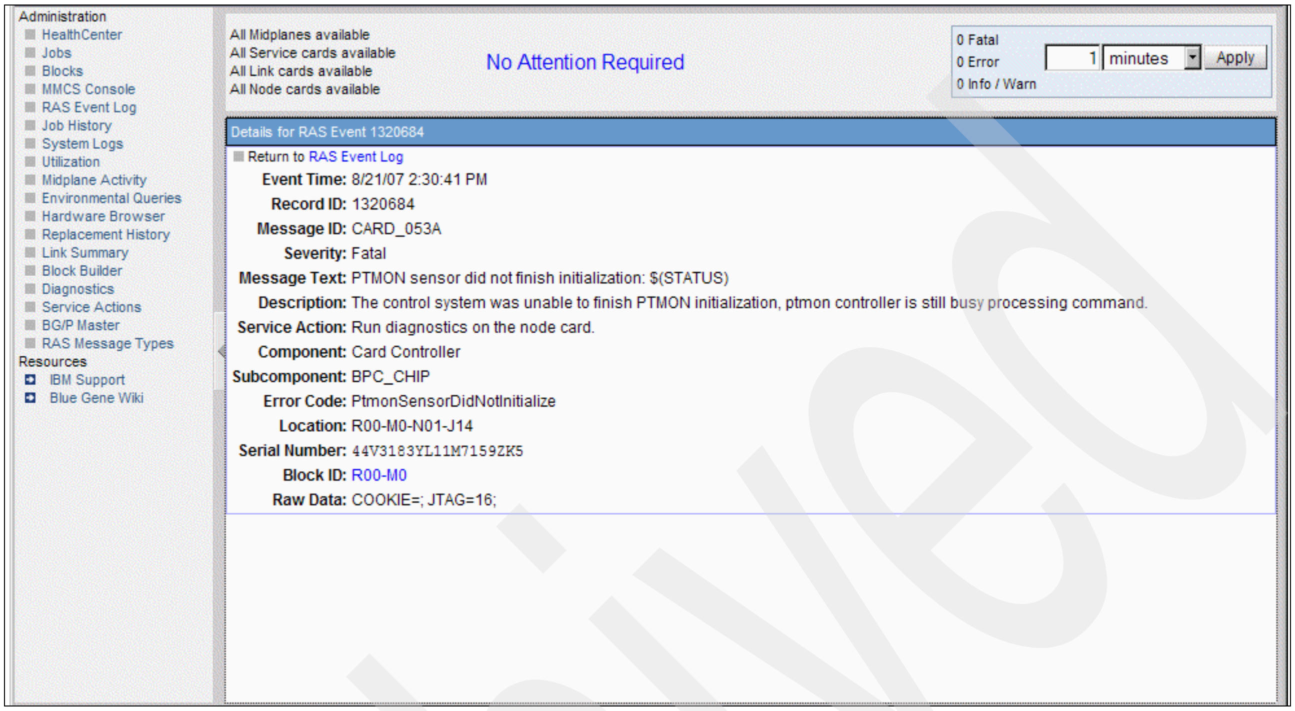
	Event Time	Message ID	Severity	Location ^	Job ID	Message
ℹ	8/21/07 2:30:41 PM	CARD_0360	Fatal	R00-M0-N01		Problem configuring PTMON facility on a processor card: rc=0xFFFFFFFB
ℹ	8/21/07 2:30:41 PM	CARD_053A	Fatal	R00-M0-N01-J14		PTMON sensor did not finish initialization: $(STATUS)
ℹ	8/21/07 2:30:41 PM	CARD_0360	Fatal	R00-M0-N09		Problem configuring PTMON facility on a processor card: rc=0xFFFFFFFB
ℹ	8/21/07 2:30:41 PM	CARD_053A	Fatal	R00-M0-N09-J17		PTMON sensor did not finish initialization: $(STATUS)
ℹ	8/21/07 2:30:41 PM	CARD_0360	Fatal	R00-M0-N14		Problem configuring PTMON facility on a processor card: rc=0xFFFFFFFB
ℹ	8/21/07 2:30:41 PM	CARD_053A	Fatal	R00-M0-N14-J19		PTMON sensor did not finish initialization: $(STATUS)
	8/21/07					Problem configuring PTMON facility on a processor card:

Figure 2-10 Click the Column head to sort

In the extreme left column of the RAS event log, there is an additional information icon, as shown in Figure 2-11.

Figure 2-11 Additional information icon

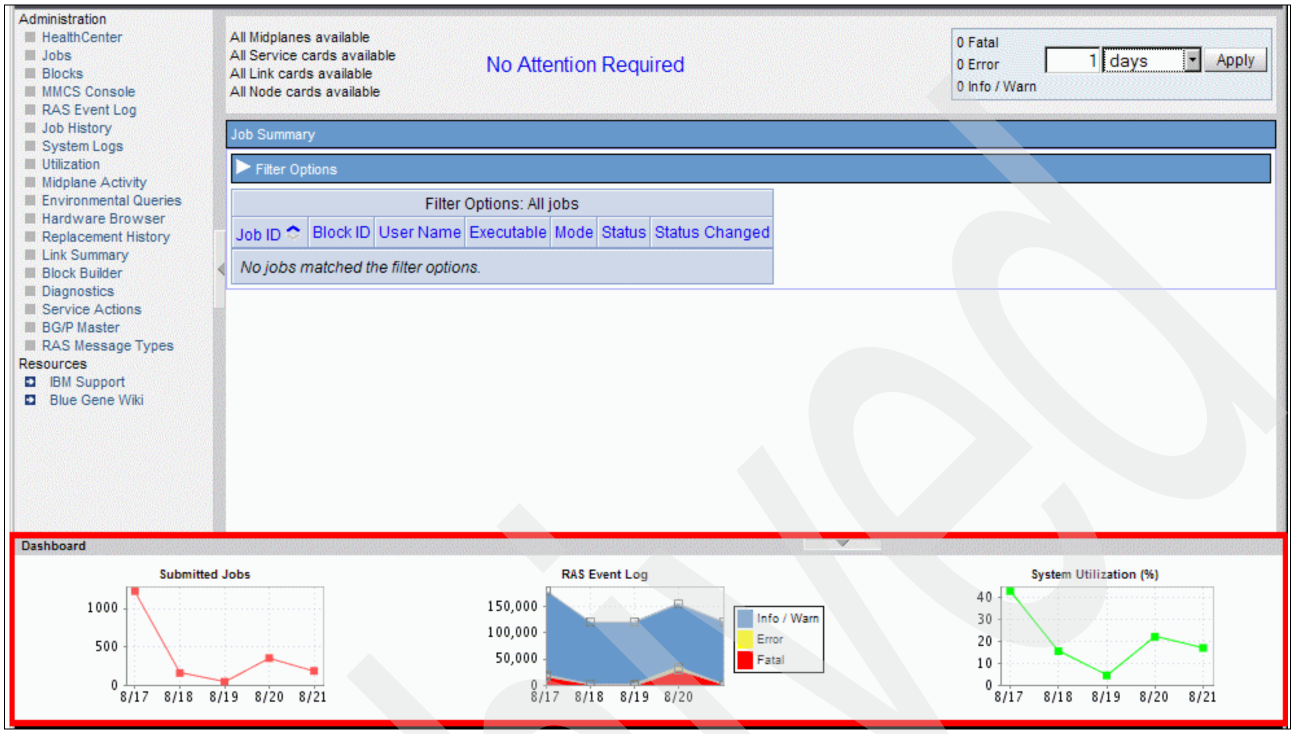
Clicking the icon displays a message details page such as the one shown in Figure 2-12. The message details contain information about the error and, in some cases, what action, if any, should be taken.



Event Time: 8/21/07 2:30:41 PM
Record ID: 1320684
Message ID: CARD_053A
Severity: Fatal
Message Text: PTMON sensor did not finish initialization: $(STATUS)
Description: The control system was unable to finish PTMON initialization, ptmon controller is still busy processing command.
Service Action: Run diagnostics on the node card.
Component: Card Controller
Subcomponent: BPC_CHIP
Error Code: PtmonSensorDidNotInitialize
Location: R00-M0-N01-J14
Serial Number: 44V3183YL11M7159ZK5
Block ID: R00-M0
Raw Data: COOKIE=; JTAG=16;

Figure 2-12 RAS event detail page

The bottom section of the Navigator pages, Figure 2-13, is referred to as the *Dashboard*. By default, the Dashboard section of the Navigator contains three graphs. The content of the graphs give you a feel for the system's utilization and general health.



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

No Attention Required

0 Fatal
0 Error
0 Info / Warn

Filter Options: All jobs

Job ID	Block ID	User Name	Executable	Mode	Status	Status Changed
No jobs matched the filter options.						

Figure 2-13 Navigator Dashboard

You can collapse the Dashboard by clicking the down arrow on the splitter bar (Figure 2-14). Clicking the arrow again restores the Dashboard.

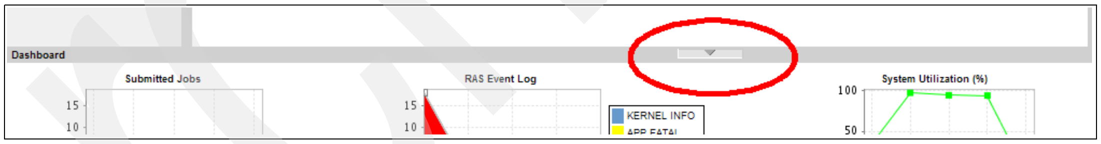


Figure 2-14 Splitter bar

Submitted Jobs chart

The Submitted Jobs chart (Figure 2-15) shows a summary of the jobs that have been submitted over the last five days. Hovering over any of the red squares in the graph gives you the date and the total of jobs counted on that day. Clicking anywhere on the chart takes you to the Job History page. You can find more details about the Job History page in 2.3.6, "Job History" on page 53.

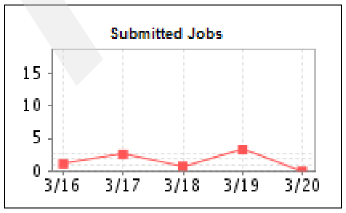


Figure 2-15 Submitted Jobs graph

RAS event graph

The Software RAS Events chart (Figure 2-16) gives you a color-coded, visual report of RAS events for the last 5 days. When you hover over any of the squares, you see a recap of the date and number of that particular type of error. Clicking any of the squares submits a query on that type of error, and you are directed to the RAS Event Log page. You can find more information about that log in 2.3.5, "RAS Event Log" on page 49.

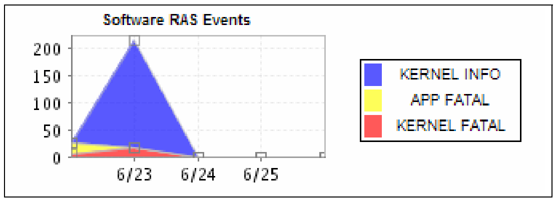


Figure 2-16 RAS graph

System Utilization graph

Finally, the System Utilization graph (Figure 2-17) shows the percentage of utilization for the date specified. As with the other charts, when you hover over one of the squares, you see the daily results. For further details about system Utilization, see 2.3.8, "Utilization" on page 58.

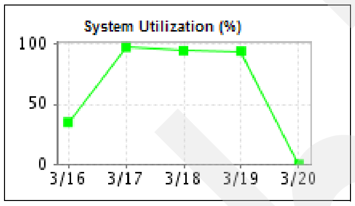


Figure 2-17 System Utilization graph

The three-graph view of the Dashboard applies to the user version of the Navigator also. The only differences is that the mouse-over function does not display any data about the System Utilization graph and the link to the Utilization page does not work.

High Throughput Computing Jobs graph

You can customize the Navigator Dashboard to show submitted High Throughput Computing Jobs (HTC) graph jobs, as well as HPC type jobs. The code to generate the graph is included in the release. To enable the HTC jobs graph, you must add the `chart-plugin` container to the local Navigator configuration file (`/bgsys/local/etc/navigator-config.xml`) by inserting the lines that are highlighted in bold in Example 2-20 into the file.

Example 2-20 Adding the chart-plugin

```
<navigator>
. . .
  <chart-plugin>
    <name>htc_jobs</name>
    <enabled>true</enabled>
  </chart-plugin>
. . .
</navigator>
```

Save the file and then restart the Navigator using the following command (as bgpadmin):

```
./bgpmaster restart navigator_server
```

Figure 2-18 shows the Dashboard with the new graph.

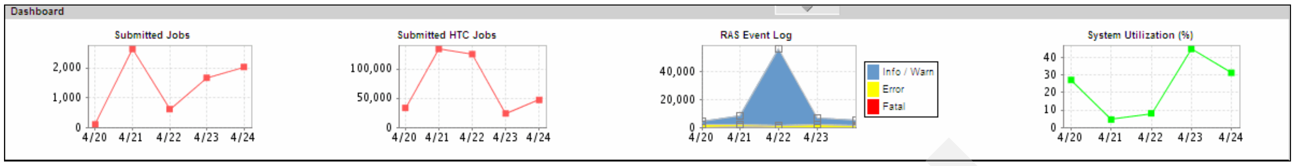


Figure 2-18 Navigator Dashboard with HTC graph

The Dashboard is another area of the Navigator that you can redefine, as described in 2.4, “Customizing Blue Gene Navigator” on page 73.

Lookup tool

The bottom section of the Navigator interface contains a *Lookup* tool and an *Auto refresh* function. The Lookup tool allows you to search for block or job information. You can search specific information, or use the Lookup to browse to other pages within the Navigator. Clicking **What can I look up?** opens the tool’s help page.

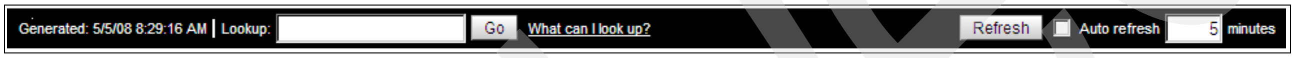


Figure 2-19 Search bar

To the right side of the search bar is the refresh function. The *Refresh* button located on this bar is the recommended method of refreshing the page. You can also set your browser to refresh automatically by supplying the value (in minutes) that you want to wait before renewing the display and then enabling the “Auto restart” option. After you have enabled Auto refresh, it continues to refresh on any Navigator page that is open.

2.2.1 User preferences and session data

Blue Gene Navigator stores user preferences by user name. If several system administrators log in to Blue Gene Navigator with the same user name, they all share the same user preferences. Therefore, we recommend that every system administrator use a different user name when logging in to Blue Gene Navigator.

Table 2-2 shows the user preferences saved by user name.

Table 2-2 User preferences saved from session to session

Item	Description
AttentionMagnitude	The interval for the RAS events in the Attention area
AttentionUnit	The type of units (seconds, minutes, hours, and days) to use when filtering RAS events shown in the attention area
AutoHighlight	Auto highlight enabled or disabled in Log Viewer
DashboardDisplayed	Whether the Dashboard is displayed or not
DiagsLogQueryPageSize	Number of rows on a page for the Diagnostics results by location
DiagsRunQueryPageSize	Number of rows on a page for the Diagnostics results by diagnostics run
Directory	Default directory for log files in System Logs
ErrcodesIntervalUnit	Interval for Error Codes event count filter option

Item	Description
ErrcodesIntervalMagnitude	Interval for Error Codes event count filter option
ErrcodesPageSize	Number of rows on a page for the Error Codes results
JobHistoryJobType	Job Type filter option in Job History
JobHistoryPageSize	Number of rows on a page for the Job History results
JobSummaryJobType	Job Type filter option in Job Summary
JobSummaryPageSize	Number of rows on a page for the Job Summary results
Lines	Number of lines to display in Log Viewer
RASEventLogPageSize	Number of rows on a page for the RAS events results
RefreshTimeout	Refresh time for page auto-refresh
ReplHistPageSize	Number of rows on a page for the Replacement History results
Timer	Auto refresh interval in Log Viewer

Blue Gene Navigator also maintains session data, such as whether the Dashboard is currently visible at the bottom of the Web page (see Figure 2-13 on page 36). If you open multiple tabs or multiple windows in the current browser session, all tabs and windows share the same session data.

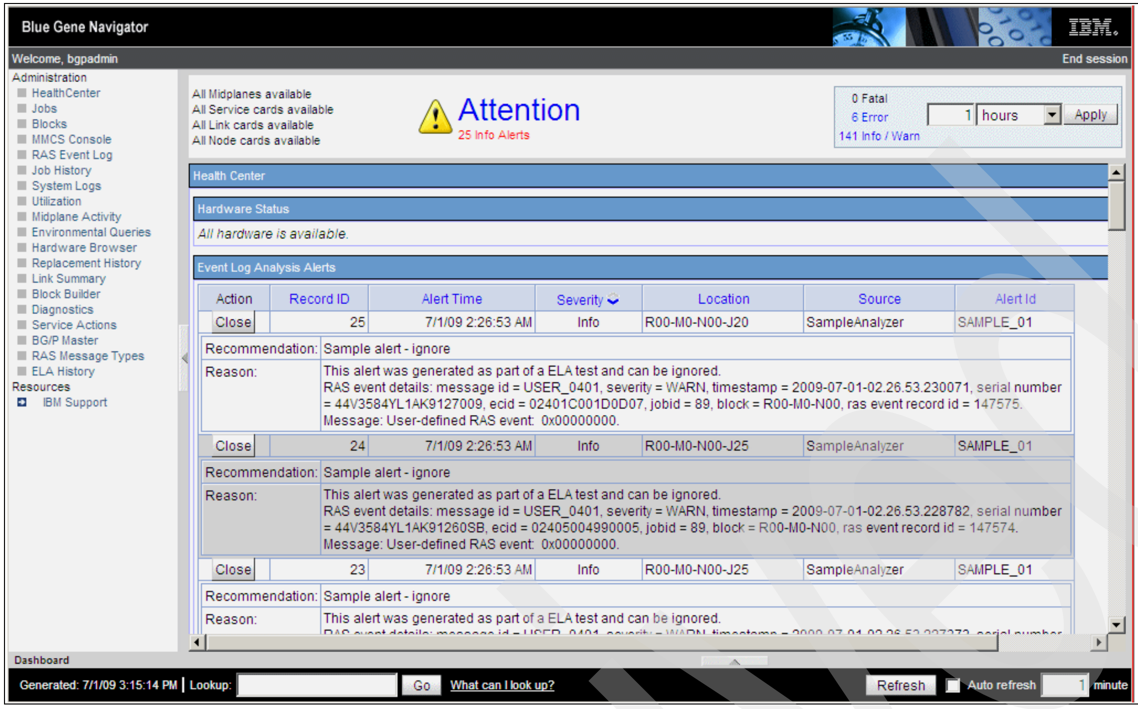
2.3 Using the Navigator

Using the Blue Gene Navigator is very intuitive. Select a link in the navigation pane, and you are taken to a page that allows you to accomplish tasks that are associated with that link. This section introduces the features that are available in the Navigator, many of which you use in the day-to-day administration of the Blue Gene/P system.

2.3.1 Health Center

We touched briefly on the Health Center in 2.1, “Blue Gene Navigator overview” on page 18. In this section, we take a closer look at the Health Center and explore its functionality. We recommend that the system administrator monitor the Navigator Health Center for conditions that require action.

Figure 2-20 shows the Health Center main page.



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

Attention
25 Info Alerts

0 Fatal
6 Error
141 Info / Warn
1 hours Apply

Health Center

Hardware Status

All hardware is available.

Event Log Analysis Alerts

Action	Record ID	Alert Time	Severity	Location	Source	Alert Id
Close	25	7/1/09 2:26:53 AM	Info	R00-M0-N00-J20	SampleAnalyzer	SAMPLE_01

Recommendation:	Sample alert - ignore
Reason:	This alert was generated as part of a ELA test and can be ignored. RAS event details: message id = USER_0401, severity = WARN, timestamp = 2009-07-01-02.26.53.230071, serial number = 44V3584YL1AK9127009, ecid = 02401C001D0D07, jobid = 89, block = R00-M0-N00, ras event record id = 147575. Message: User-defined RAS event: 0x00000000.

Action	Record ID	Alert Time	Severity	Location	Source	Alert Id
Close	24	7/1/09 2:26:53 AM	Info	R00-M0-N00-J25	SampleAnalyzer	SAMPLE_01

Recommendation:	Sample alert - ignore
Reason:	This alert was generated as part of a ELA test and can be ignored. RAS event details: message id = USER_0401, severity = WARN, timestamp = 2009-07-01-02.26.53.228782, serial number = 44V3584YL1AK91260SB, ecid = 02405004990005, jobid = 89, block = R00-M0-N00, ras event record id = 147574. Message: User-defined RAS event: 0x00000000.

Action	Record ID	Alert Time	Severity	Location	Source	Alert Id
Close	23	7/1/09 2:26:53 AM	Info	R00-M0-N00-J25	SampleAnalyzer	SAMPLE_01

Recommendation:	Sample alert - ignore
Reason:	This alert was generated as part of a ELA test and can be ignored.

Generated: 7/1/09 3:15:14 PM | Lookup: Go What can I look up? Refresh Auto refresh 1 minute

Figure 2-20 Health Center home

The content area of the main page includes the following subheadings:

- Hardware Status
- Event Log Analysis Alerts
- Recent Jobs with Fatal RAS Events
- Running Jobs with Proactive Power Management
- Running Jobs with Reactive Power Management
- Recent Fatal RAS Events
- Attention Messages

Important: The Event Log Analysis Alerts tool requires configuration. See Chapter 8, “Event Log Analysis” on page 155 for information about setting up the tool.

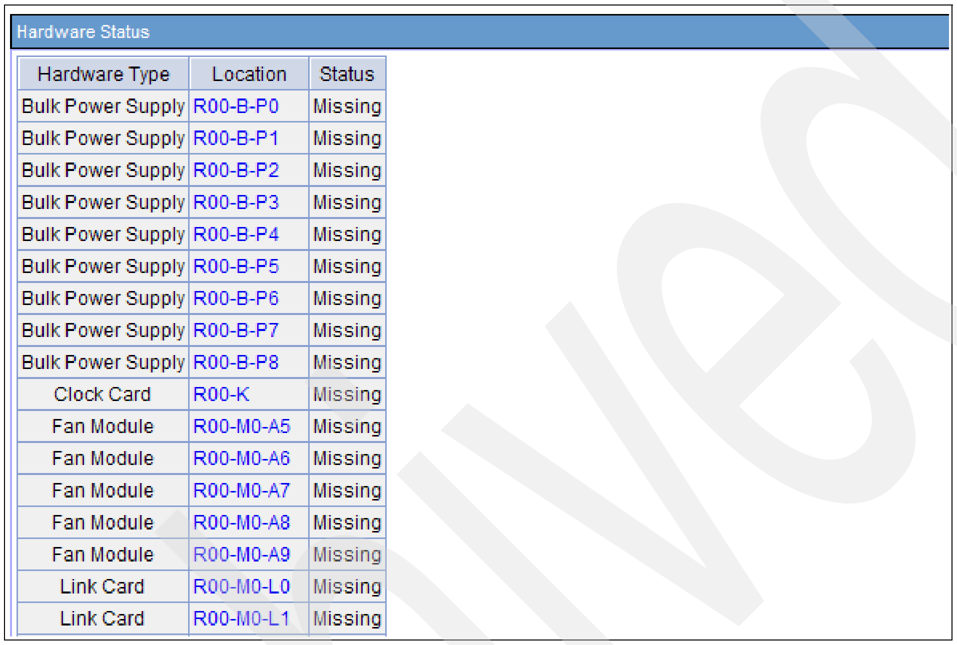
Also, the Attention Messages subheading is dynamic. It only displays in the Health Center if you have customized the Attention plug-in as described in 2.4, “Customizing Blue Gene Navigator” on page 73.

Hardware status

When there are no current hardware issues with the Blue Gene/P hardware, this message box returns the following message:

```
All hardware is available.
```

However, if hardware is missing or somehow in error, the Navigator shows a message similar to that shown in Figure 2-21, which supplies the hardware type, location, and status.

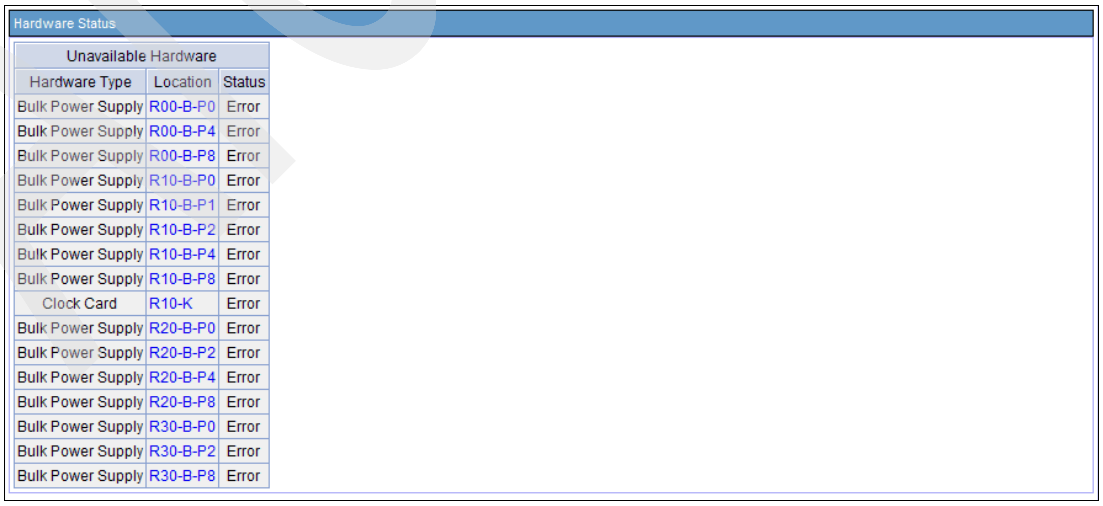


Hardware Status

Hardware Type	Location	Status
Bulk Power Supply	R00-B-P0	Missing
Bulk Power Supply	R00-B-P1	Missing
Bulk Power Supply	R00-B-P2	Missing
Bulk Power Supply	R00-B-P3	Missing
Bulk Power Supply	R00-B-P4	Missing
Bulk Power Supply	R00-B-P5	Missing
Bulk Power Supply	R00-B-P6	Missing
Bulk Power Supply	R00-B-P7	Missing
Bulk Power Supply	R00-B-P8	Missing
Clock Card	R00-K	Missing
Fan Module	R00-M0-A5	Missing
Fan Module	R00-M0-A6	Missing
Fan Module	R00-M0-A7	Missing
Fan Module	R00-M0-A8	Missing
Fan Module	R00-M0-A9	Missing
Link Card	R00-M0-L0	Missing
Link Card	R00-M0-L1	Missing

Figure 2-21 Health Center hardware status window

Hardware in the *Error* state typically needs to be replaced. A subset of RAS events causes hardware to be put in the Error state when the condition is considered unrecoverable and likely to fail repeatedly. Job schedulers, such as Load Leveler, avoid dispatching jobs to hardware that is in an Error state. Figure 2-22 shows the Hardware Status portion of the Health Center when there is hardware in an Error state.



Hardware Status

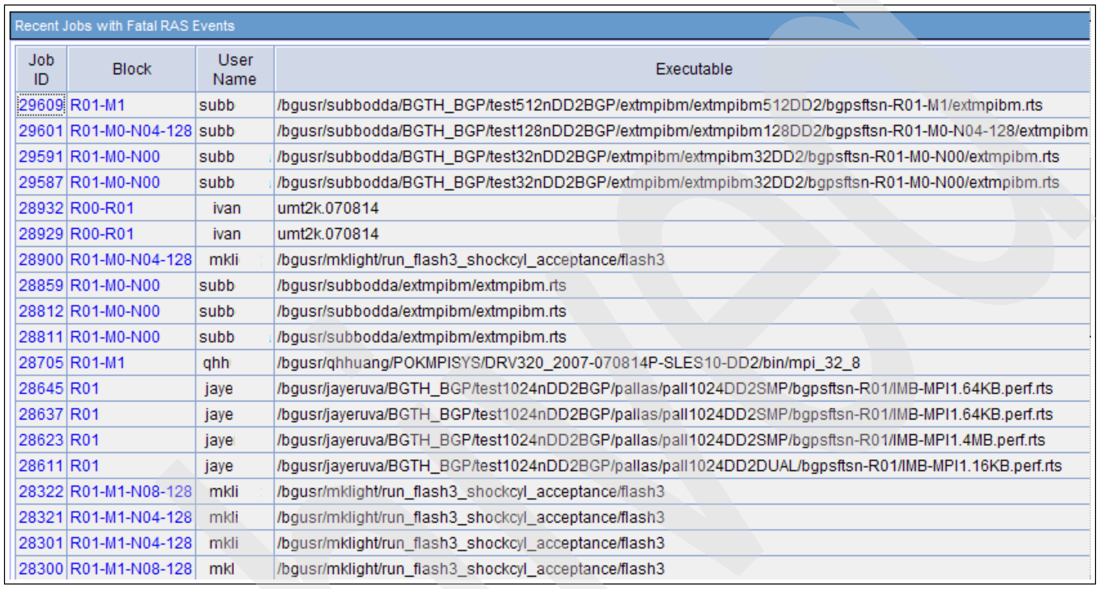
Unavailable Hardware		
Hardware Type	Location	Status
Bulk Power Supply	R00-B-P0	Error
Bulk Power Supply	R00-B-P4	Error
Bulk Power Supply	R00-B-P8	Error
Bulk Power Supply	R10-B-P0	Error
Bulk Power Supply	R10-B-P1	Error
Bulk Power Supply	R10-B-P2	Error
Bulk Power Supply	R10-B-P4	Error
Bulk Power Supply	R10-B-P8	Error
Clock Card	R10-K	Error
Bulk Power Supply	R20-B-P0	Error
Bulk Power Supply	R20-B-P2	Error
Bulk Power Supply	R20-B-P4	Error
Bulk Power Supply	R20-B-P8	Error
Bulk Power Supply	R30-B-P0	Error
Bulk Power Supply	R30-B-P2	Error
Bulk Power Supply	R30-B-P8	Error

Figure 2-22 Hardware in error

Recent jobs with fatal RAS events

This section of the Health Center reports all of the jobs that ran in the previous 24 hours (the default) that reported an RAS event. As shown in Figure 2-23, the report includes information about, and a link to, the job and block. The table also shows the user, executable, time of, and the message that is associated with the event.

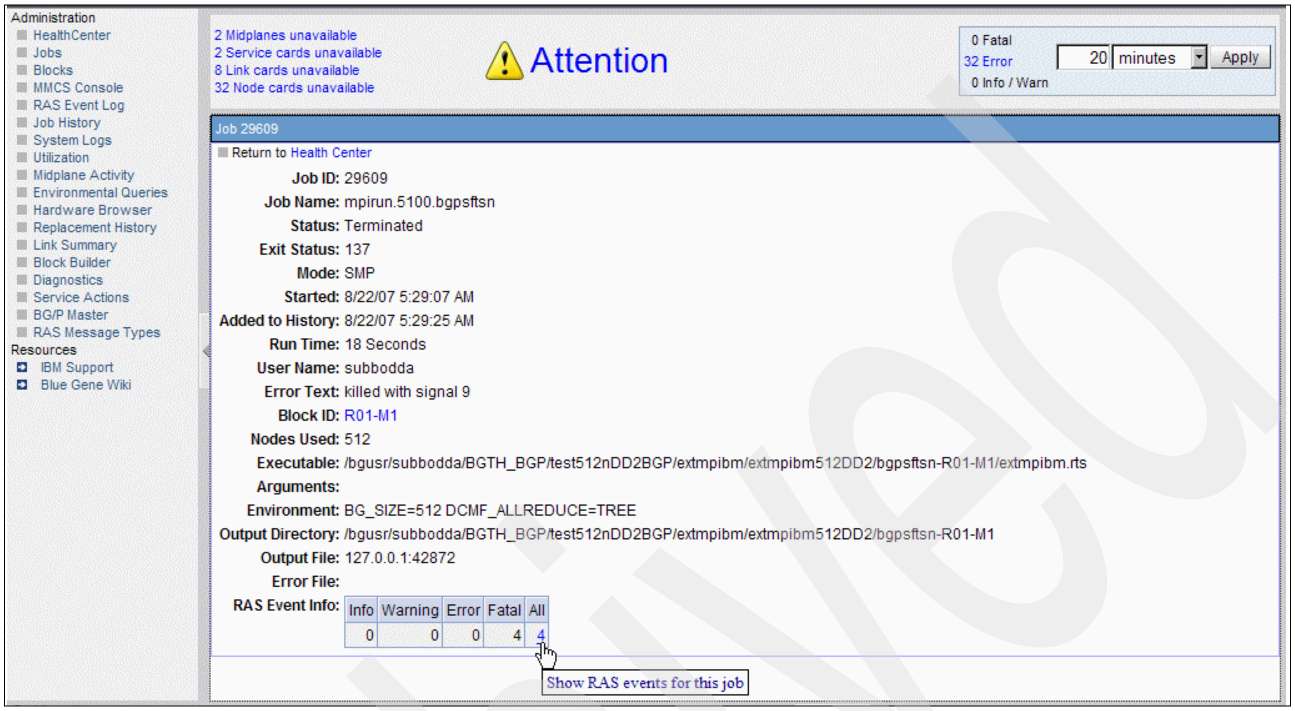
Jobs that encounter Fatal RAS Events from the compute node or Linux Kernels (RAS `id=KERN_XXXX`) result in terminating the job and freeing the block. A subset of these events marks the hardware in an Error state, which indicates that part replacement is recommended.


Recent Jobs with Fatal RAS Events

Job ID	Block	User Name	Executable
29609	R01-M1	subb	/bgusr/subbodda/BGTH_BGP/test512nDD2BGP/extmpibm/extmpibm512DD2/bgpsftsn-R01-M1/extmpibm.rts
29601	R01-M0-N04-128	subb	/bgusr/subbodda/BGTH_BGP/test128nDD2BGP/extmpibm/extmpibm128DD2/bgpsftsn-R01-M0-N04-128/extmpibm
29591	R01-M0-N00	subb	/bgusr/subbodda/BGTH_BGP/test32nDD2BGP/extmpibm/extmpibm32DD2/bgpsftsn-R01-M0-N00/extmpibm.rts
29587	R01-M0-N00	subb	/bgusr/subbodda/BGTH_BGP/test32nDD2BGP/extmpibm/extmpibm32DD2/bgpsftsn-R01-M0-N00/extmpibm.rts
28932	R00-R01	ivan	umt2k.070814
28929	R00-R01	ivan	umt2k.070814
28900	R01-M0-N04-128	mkli	/bgusr/mklight/run_flash3_shockcyl_acceptance/flash3
28859	R01-M0-N00	subb	/bgusr/subbodda/extmpibm/extmpibm.rts
28812	R01-M0-N00	subb	/bgusr/subbodda/extmpibm/extmpibm.rts
28811	R01-M0-N00	subb	/bgusr/subbodda/extmpibm/extmpibm.rts
28705	R01-M1	qhh	/bgusr/qhhuang/POKMPISYS/DRV320_2007-070814P-SLES10-DD2/bin/mpi_32_8
28645	R01	jaye	/bgusr/jayeruva/BGTH_BGP/test1024nDD2BGP/pallas/pall1024DD2SMP/bgpsftsn-R01/IMB-MPI1.64KB.perf.rts
28637	R01	jaye	/bgusr/jayeruva/BGTH_BGP/test1024nDD2BGP/pallas/pall1024DD2SMP/bgpsftsn-R01/IMB-MPI1.64KB.perf.rts
28623	R01	jaye	/bgusr/jayeruva/BGTH_BGP/test1024nDD2BGP/pallas/pall1024DD2SMP/bgpsftsn-R01/IMB-MPI1.4MB.perf.rts
28611	R01	jaye	/bgusr/jayeruva/BGTH_BGP/test1024nDD2BGP/pallas/pall1024DD2DUAL/bgpsftsn-R01/IMB-MPI1.16KB.perf.rts
28322	R01-M1-N08-128	mkli	/bgusr/mklight/run_flash3_shockcyl_acceptance/flash3
28321	R01-M1-N04-128	mkli	/bgusr/mklight/run_flash3_shockcyl_acceptance/flash3
28301	R01-M1-N04-128	mkli	/bgusr/mklight/run_flash3_shockcyl_acceptance/flash3
28300	R01-M1-N08-128	mkl	/bgusr/mklight/run_flash3_shockcyl_acceptance/flash3

Figure 2-23 Jobs with RAS events

To drill down to the actual RAS event, click the job number in the far left column to open the Job Details page shown in Figure 2-24. At the bottom of the page, there is a table that contains the number of each type RAS message that is generated by this job.



2 Midplanes unavailable
2 Service cards unavailable
8 Link cards unavailable
32 Node cards unavailable

0 Fatal
32 Error
0 Info / Warn
20 minutes

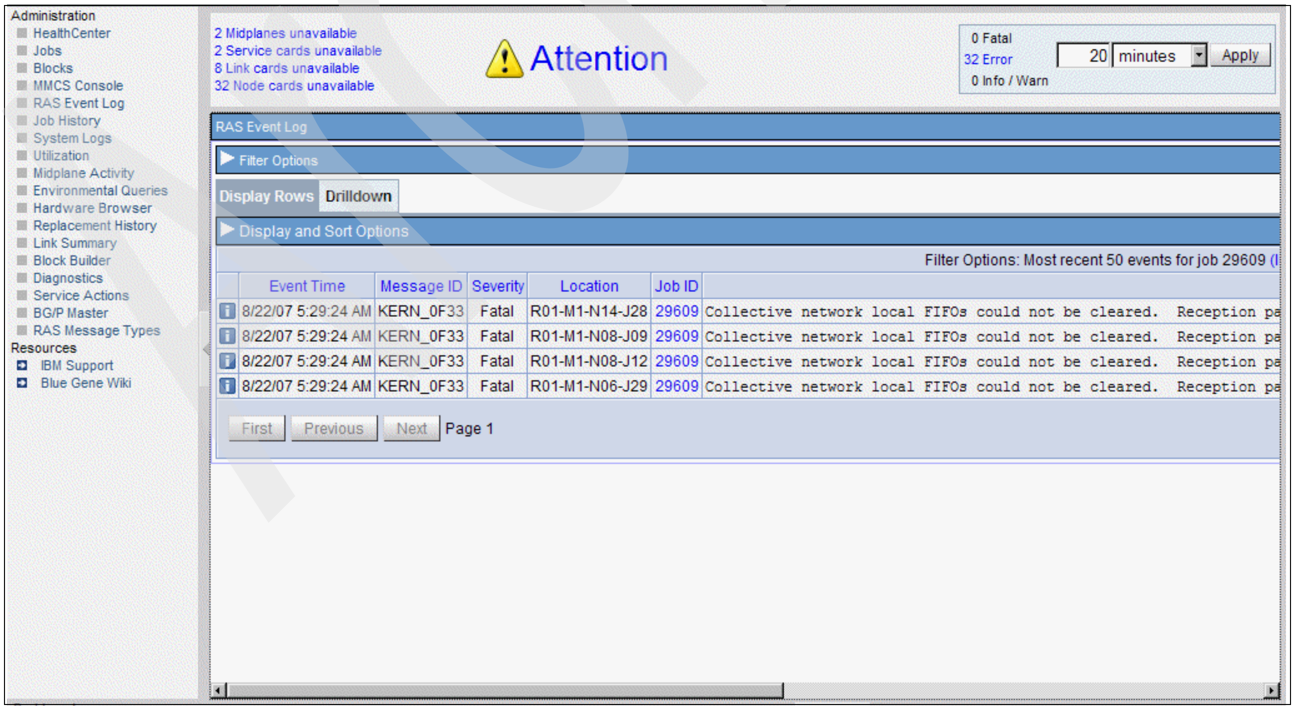
Job 29609
Return to Health Center

Job ID: 29609
Job Name: mpirun.5100.bgpsftsn
Status: Terminated
Exit Status: 137
Mode: SMP
Started: 8/22/07 5:29:07 AM
Added to History: 8/22/07 5:29:25 AM
Run Time: 18 Seconds
User Name: subbodda
Error Text: killed with signal 9
Block ID: R01-M1
Nodes Used: 512
Executable: /bgusr/subbodda/BGTH_BGP/test512nDD2BGP/extmpibm/extmpibm512DD2/bgpsftsn-R01-M1/extmpibm.rts
Arguments:
Environment: BG_SIZE=512 DCMF_ALLREDUCE=TREE
Output Directory: /bgusr/subbodda/BGTH_BGP/test512nDD2BGP/extmpibm/extmpibm512DD2/bgpsftsn-R01-M1
Output File: 127.0.0.1:42872
Error File:
RAS Event Info:

Info	Warning	Error	Fatal	All
0	0	0	4	4

Figure 2-24 Health Center Job details page

The last column of the table, labelled *All*, contains a link to the RAS database. Clicking the number in the cell that is located below *All* sends a query to the database and returns all the RAS events for the job (Figure 2-25).



2 Midplanes unavailable
2 Service cards unavailable
8 Link cards unavailable
32 Node cards unavailable

0 Fatal
32 Error
0 Info / Warn
20 minutes

RAS Event Log

Filter Options: Most recent 50 events for job 29609 (I

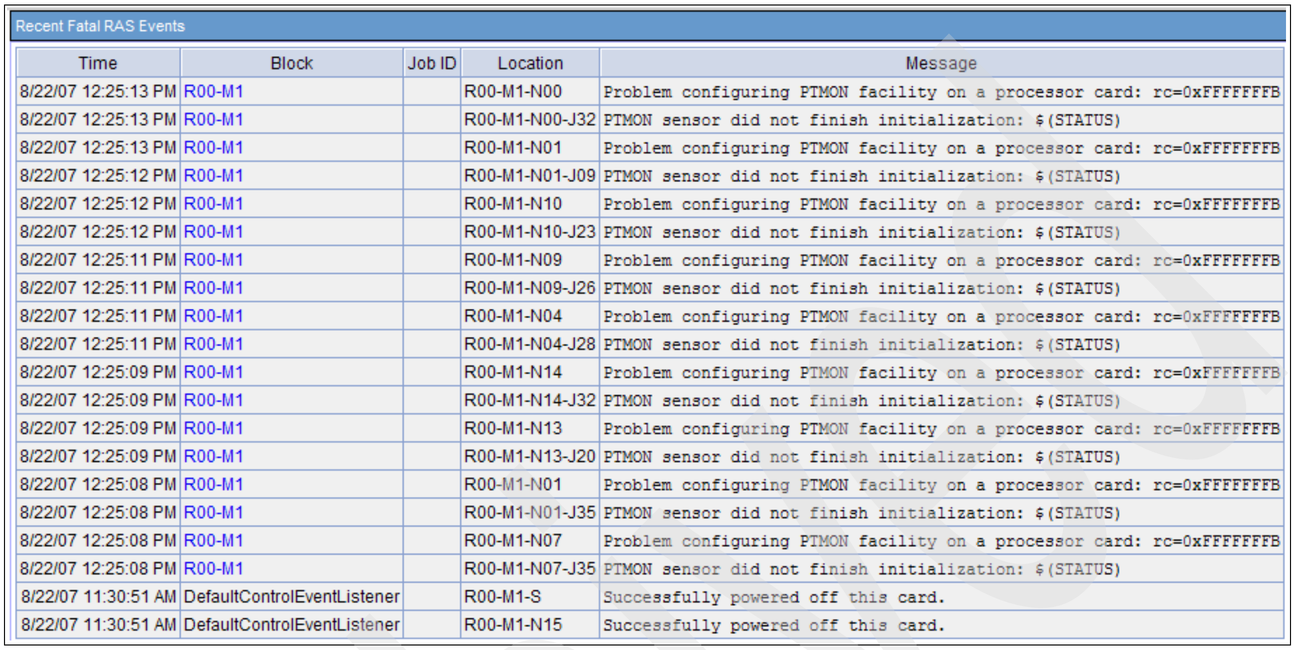
Event Time	Message ID	Severity	Location	Job ID	
8/22/07 5:29:24 AM	KERN_0F33	Fatal	R01-M1-N14-J28	29609	Collective network local FIFOs could not be cleared. Reception pa
8/22/07 5:29:24 AM	KERN_0F33	Fatal	R01-M1-N08-J09	29609	Collective network local FIFOs could not be cleared. Reception pa
8/22/07 5:29:24 AM	KERN_0F33	Fatal	R01-M1-N08-J12	29609	Collective network local FIFOs could not be cleared. Reception pa
8/22/07 5:29:24 AM	KERN_0F33	Fatal	R01-M1-N06-J29	29609	Collective network local FIFOs could not be cleared. Reception pa

Page 1

Figure 2-25 RAS query results

Recent fatal RAS events

This section of the Health Center reports the time, location, and description of fatal errors that occur on the system (Figure 2-26). The *Block* column provides a link to the block that contains the hardware that reported the error.



Recent Fatal RAS Events

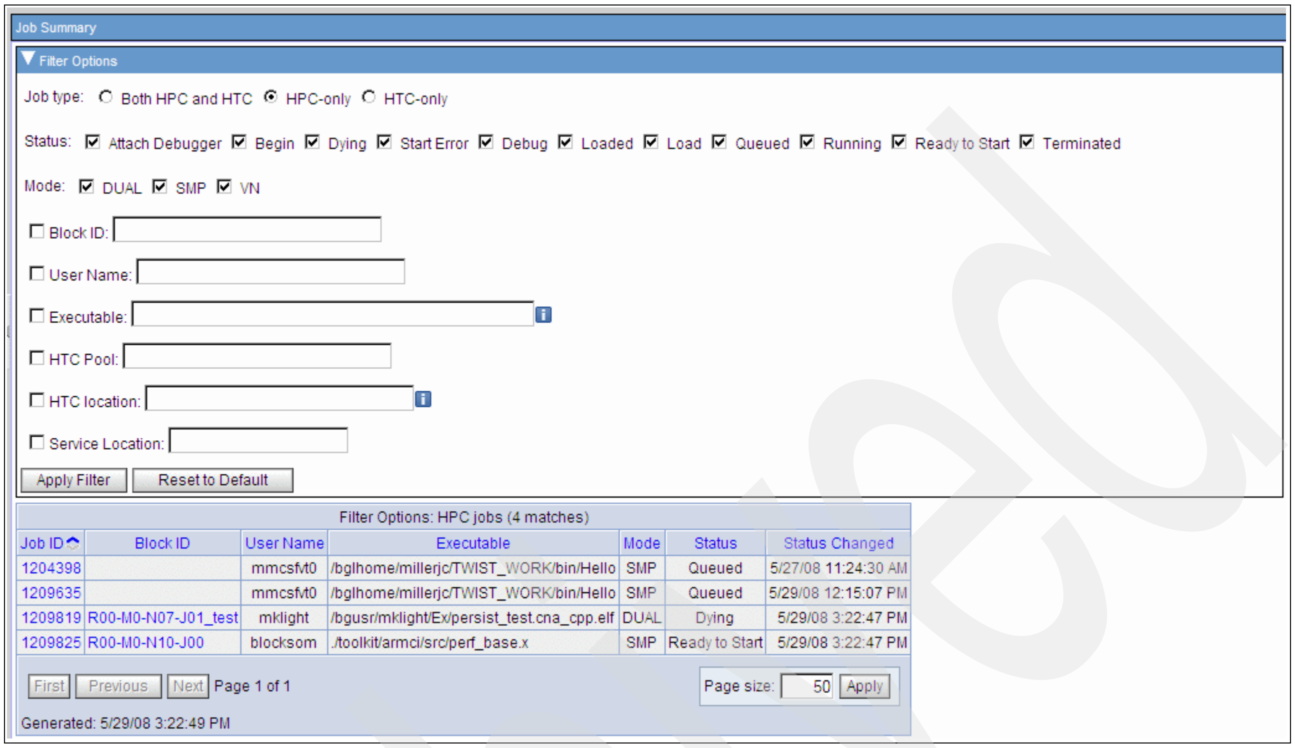
Time	Block	Job ID	Location	Message
8/22/07 12:25:13 PM	R00-M1		R00-M1-N00	Problem configuring PTMON facility on a processor card: rc=0xFFFFFFFB
8/22/07 12:25:13 PM	R00-M1		R00-M1-N00-J32	PTMON sensor did not finish initialization: $(STATUS)
8/22/07 12:25:13 PM	R00-M1		R00-M1-N01	Problem configuring PTMON facility on a processor card: rc=0xFFFFFFFB
8/22/07 12:25:12 PM	R00-M1		R00-M1-N01-J09	PTMON sensor did not finish initialization: $(STATUS)
8/22/07 12:25:12 PM	R00-M1		R00-M1-N10	Problem configuring PTMON facility on a processor card: rc=0xFFFFFFFB
8/22/07 12:25:12 PM	R00-M1		R00-M1-N10-J23	PTMON sensor did not finish initialization: $(STATUS)
8/22/07 12:25:11 PM	R00-M1		R00-M1-N09	Problem configuring PTMON facility on a processor card: rc=0xFFFFFFFB
8/22/07 12:25:11 PM	R00-M1		R00-M1-N09-J26	PTMON sensor did not finish initialization: $(STATUS)
8/22/07 12:25:11 PM	R00-M1		R00-M1-N04	Problem configuring PTMON facility on a processor card: rc=0xFFFFFFFB
8/22/07 12:25:11 PM	R00-M1		R00-M1-N04-J28	PTMON sensor did not finish initialization: $(STATUS)
8/22/07 12:25:09 PM	R00-M1		R00-M1-N14	Problem configuring PTMON facility on a processor card: rc=0xFFFFFFFB
8/22/07 12:25:09 PM	R00-M1		R00-M1-N14-J32	PTMON sensor did not finish initialization: $(STATUS)
8/22/07 12:25:09 PM	R00-M1		R00-M1-N13	Problem configuring PTMON facility on a processor card: rc=0xFFFFFFFB
8/22/07 12:25:09 PM	R00-M1		R00-M1-N13-J20	PTMON sensor did not finish initialization: $(STATUS)
8/22/07 12:25:08 PM	R00-M1		R00-M1-N01	Problem configuring PTMON facility on a processor card: rc=0xFFFFFFFB
8/22/07 12:25:08 PM	R00-M1		R00-M1-N01-J35	PTMON sensor did not finish initialization: $(STATUS)
8/22/07 12:25:08 PM	R00-M1		R00-M1-N07	Problem configuring PTMON facility on a processor card: rc=0xFFFFFFFB
8/22/07 12:25:08 PM	R00-M1		R00-M1-N07-J35	PTMON sensor did not finish initialization: $(STATUS)
8/22/07 11:30:51 AM	DefaultControlEventListener		R00-M1-S	Successfully powered off this card.
8/22/07 11:30:51 AM	DefaultControlEventListener		R00-M1-N15	Successfully powered off this card.

Figure 2-26 Fatal errors reported in Health Center

2.3.2 Jobs

The Jobs link shows a summary of jobs that are currently running or that are in the queue to run. With the introduction of HTC, there might be thousands of jobs in the Blue Gene database, which can be overwhelming for the user and can take a long time to display. So, the Navigator allows for many filtering options and paging of the running jobs.

Figure 2-27 shows all of the options that are available to filter the view of jobs that are currently running on the system or that are in the queue waiting to run.



Job Summary

Filter Options

Job type: Both HPC and HTC [✓ HPC-only] HTC-only

Status: [✓ Attach Debugger] [✓ Begin] [✓ Dying] [✓ Start Error] [✓ Debug] [✓ Loaded] [✓ Load] [✓ Queued] [✓ Running] [✓ Ready to Start] [✓ Terminated]

Mode: [✓ DUAL] [✓ SMP] [✓ VN]

Block ID:

User Name:

Executable:

HTC Pool:

HTC location:

Service Location:

Filter Options: HPC jobs (4 matches)

Job ID	Block ID	User Name	Executable	Mode	Status	Status Changed
1204398		mmcsfvt0	/bglhome/millerjc/TWIST_WORK/bin/Hello	SMP	Queued	5/27/08 11:24:30 AM
1209635		mmcsfvt0	/bglhome/millerjc/TWIST_WORK/bin/Hello	SMP	Queued	5/29/08 12:15:07 PM
1209819	R00-M0-N07-J01_test	mklight	/bgusr/mklight/Ex/persist_test.cna_cpp.elf	DUAL	Dying	5/29/08 3:22:47 PM
1209825	R00-M0-N10-J00	blocksom	./toolkit/armci/src/perf_base.x	SMP	Ready to Start	5/29/08 3:22:47 PM

Page 1 of 1

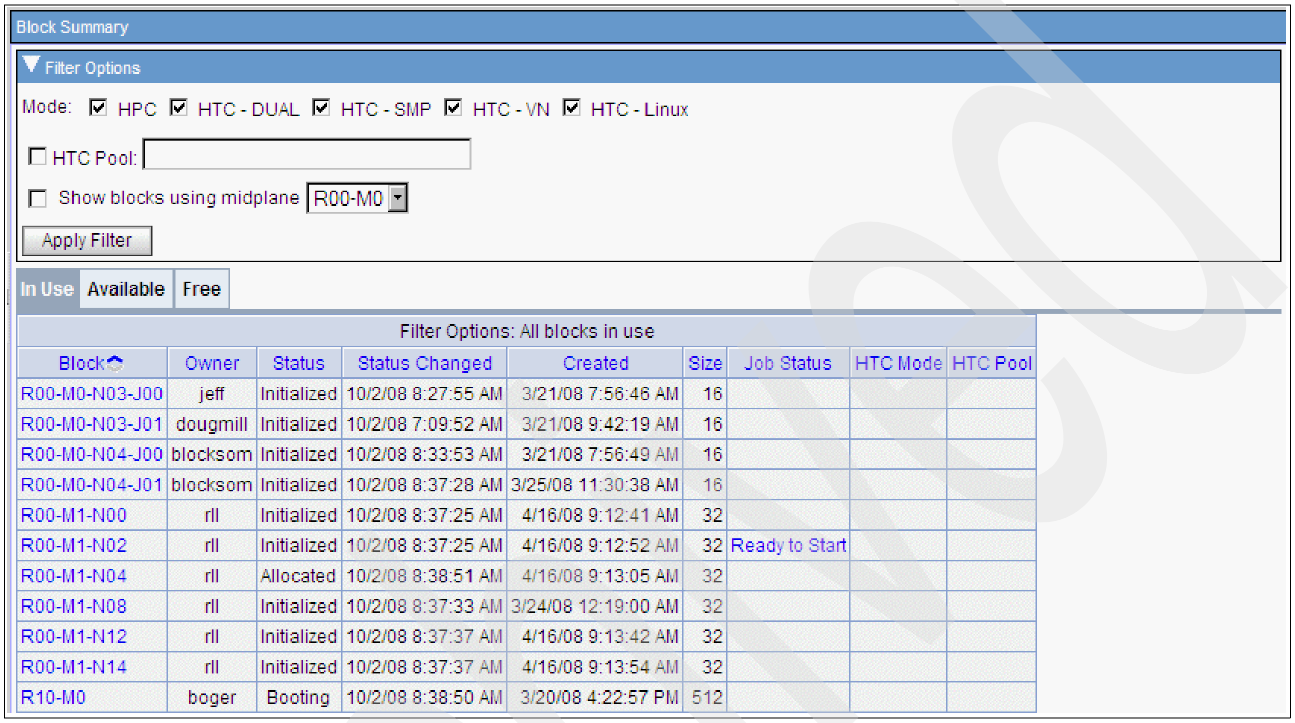
Page size: 50

Generated: 5/29/08 3:22:49 PM

Figure 2-27 Navigator Jobs page

2.3.3 Blocks

The Blocks section of the Navigator gives the option of viewing blocks that are In Use, Available, or Free (Figure 2-28). You can see the blocks in any one of the categories by clicking the appropriate tab. By default, the page contains all blocks that fit the description on the tab, on all midplanes in your system. The Filter Options allows you to narrow the results that display in each section to a single midplane, to a mode, or by HTC pool.


Block Summary
Filter Options
Mode: ☑ HPC ☑ HTC - DUAL ☑ HTC - SMP ☑ HTC - VN ☑ HTC - Linux
☐ HTC Pool:
☐ Show blocks using midplane R00-M0
Apply Filter
In Use
Available
Free

Filter Options: All blocks in use

Block	Owner	Status	Status Changed	Created	Size	Job Status	HTC Mode	HTC Pool
R00-M0-N03-J00	jeff	Initialized	10/2/08 8:27:55 AM	3/21/08 7:56:46 AM	16			
R00-M0-N03-J01	dougmill	Initialized	10/2/08 7:09:52 AM	3/21/08 9:42:19 AM	16			
R00-M0-N04-J00	blocksom	Initialized	10/2/08 8:33:53 AM	3/21/08 7:56:49 AM	16			
R00-M0-N04-J01	blocksom	Initialized	10/2/08 8:37:28 AM	3/25/08 11:30:38 AM	16			
R00-M1-N00	rll	Initialized	10/2/08 8:37:25 AM	4/16/08 9:12:41 AM	32			
R00-M1-N02	rll	Initialized	10/2/08 8:37:25 AM	4/16/08 9:12:52 AM	32	Ready to Start		
R00-M1-N04	rll	Allocated	10/2/08 8:38:51 AM	4/16/08 9:13:05 AM	32			
R00-M1-N08	rll	Initialized	10/2/08 8:37:33 AM	3/24/08 12:19:00 AM	32			
R00-M1-N12	rll	Initialized	10/2/08 8:37:37 AM	4/16/08 9:13:42 AM	32			
R00-M1-N14	rll	Initialized	10/2/08 8:37:37 AM	4/16/08 9:13:54 AM	32			
R10-M0	boger	Booting	10/2/08 8:38:50 AM	3/20/08 4:22:57 PM	512			

Figure 2-28 Block Summary page

2.3.4 Midplane Management Control System

Figure 2-29 shows the Blue Gene Navigator version of the Midplane Management Control System (MMCS) console. The MMCS console portion of Blue Gene Navigator provides nearly the same functionality as the mmcs_db_console shell. The interface supplies a text box to enter the MMCS commands. The response is returned in the window that is located below the command line.

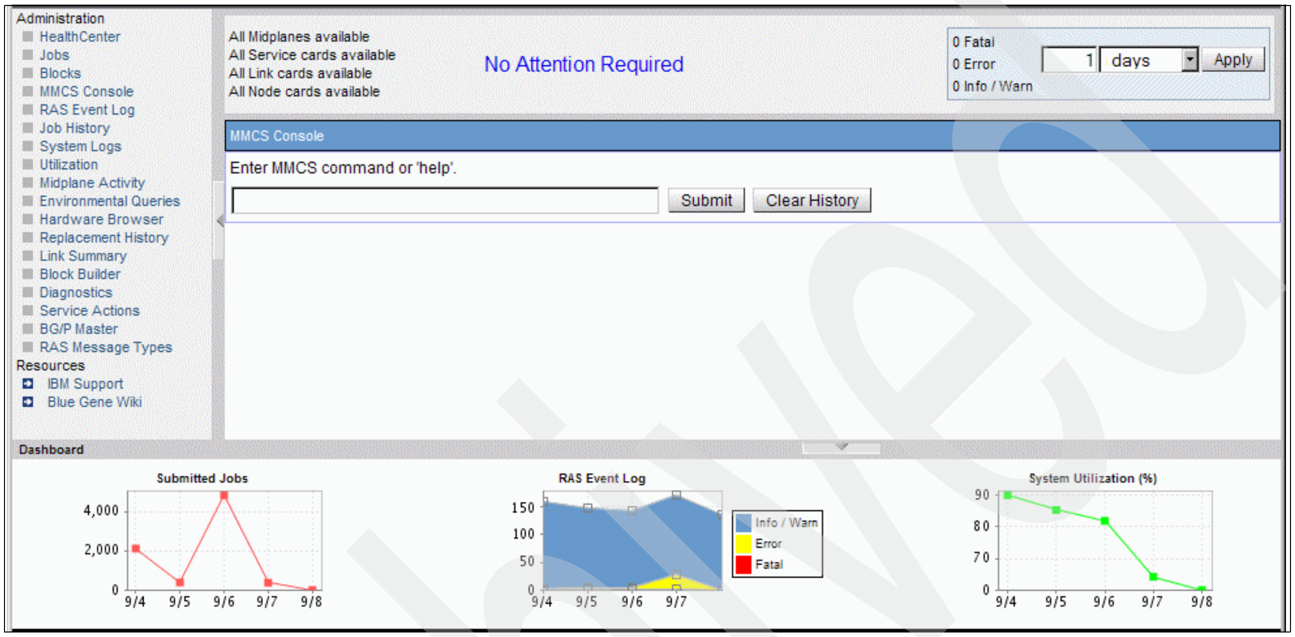
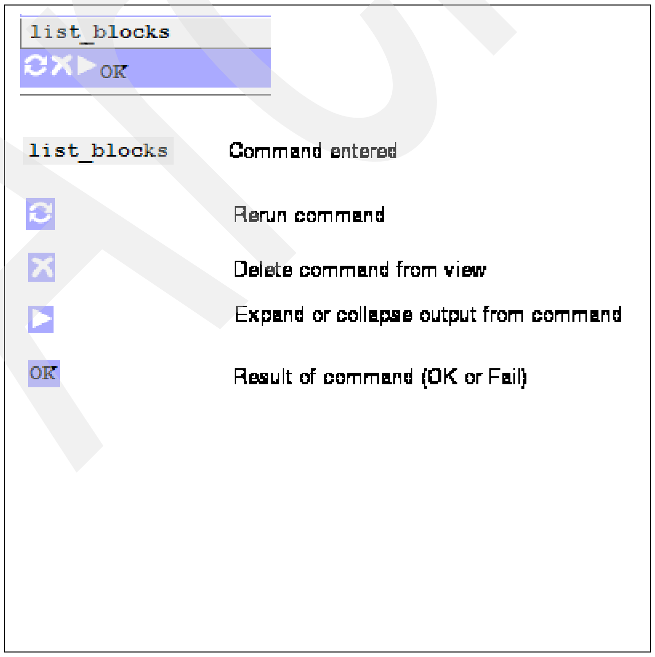


Figure 2-29 MMCS console

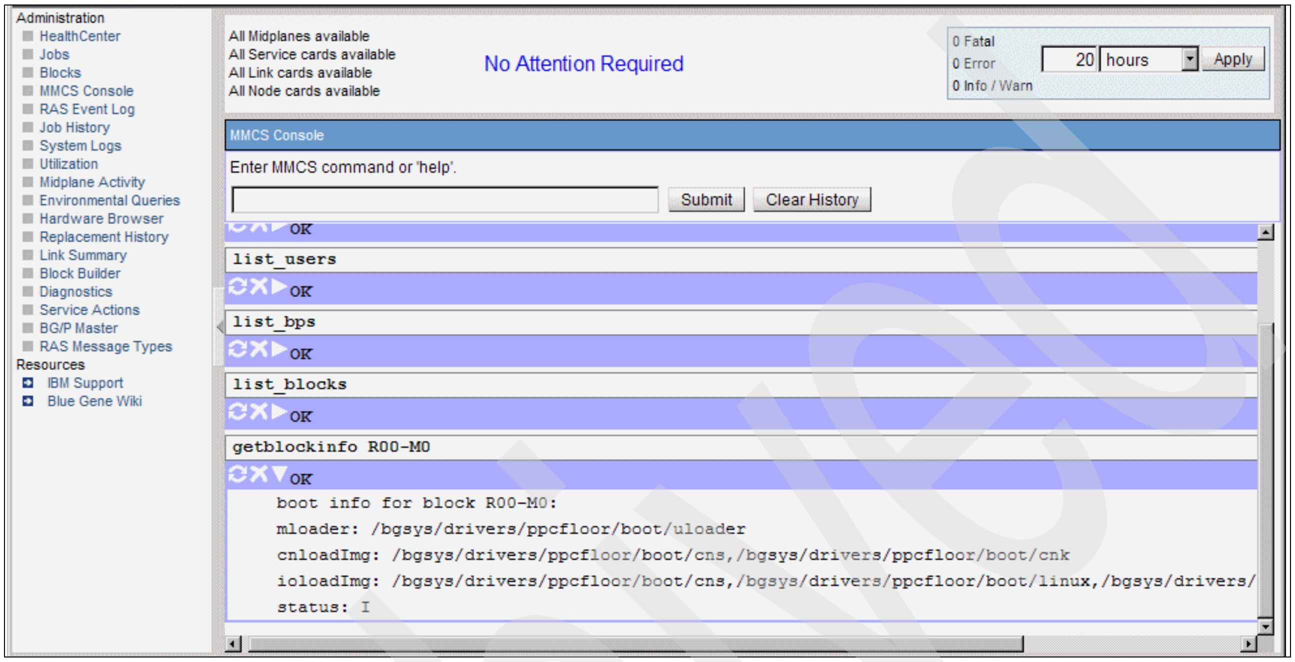
Each command's output is printed at the bottom of the panel and is associated with its own arrow, status line, and command-recycle icon. Figure 2-30 provides a legend of the icons that you see on the MMCS console page.



list_blocks	Command entered
🔄	Rerun command
✖	Delete command from view
▶	Expand or collapse output from command
OK	Result of command (OK or Fail)

Figure 2-30 MMCS console legend

For this example, we ran several **`list`** commands followed by a **`getblockinfo`** command. You can use the scroll bar, to the right of the main content area, to see the commands run and to see the output during the current session. The Navigator displays the most recent command and its output at the bottom of the panel. When a new command is executed the output from the previous request is collapsed. See Figure 2-31.



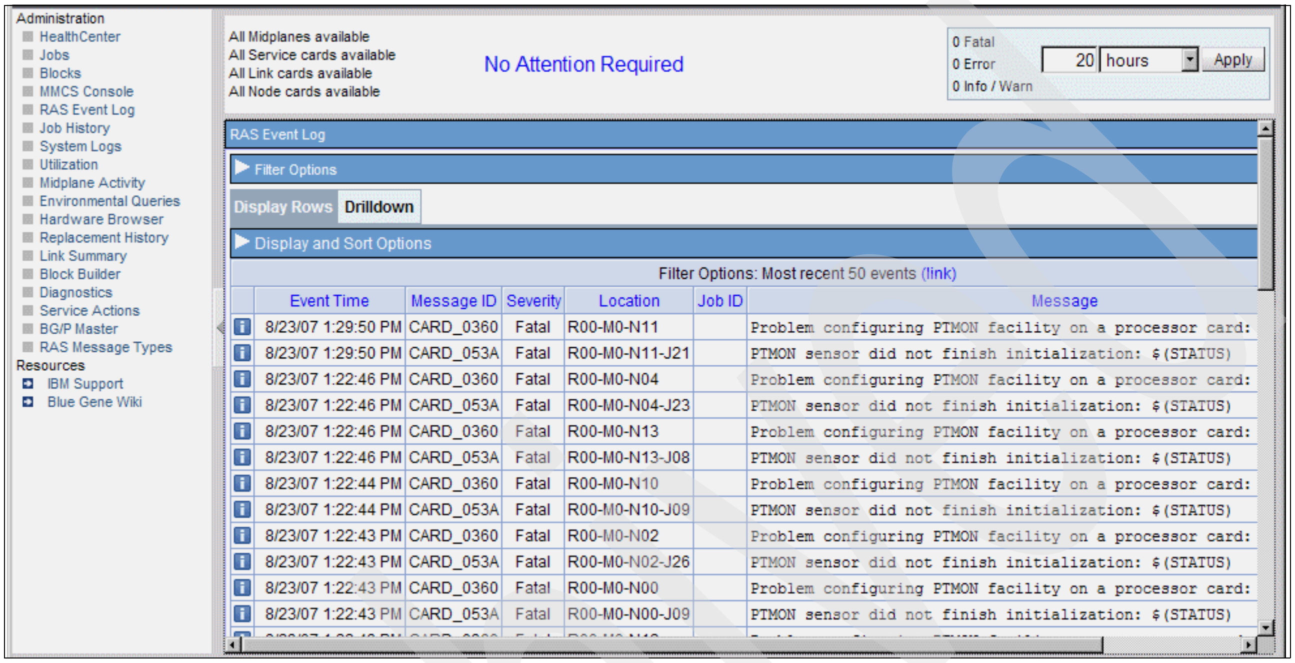
```
OK
list_users
OK
list_bps
OK
list_blocks
OK
getblockinfo R00-M0
OK
     boot info for block R00-M0:
     mloader: /bgsys/drivers/ppcfloor/boot/uloader
     cnloadImg: /bgsys/drivers/ppcfloor/boot/cns,/bgsys/drivers/ppcfloor/boot/cnk
     ioloadImg: /bgsys/drivers/ppcfloor/boot/cns,/bgsys/drivers/ppcfloor/boot/linux,/bgsys/drivers/
     status: I
```

Figure 2-31 Navigator's MMCS interface

For more specific information about the MMCS server and MMCS console, see Chapter 5, "Midplane Management Control System" on page 115.

2.3.5 RAS Event Log

The RAS Event Log page offers various ways to view system events that generated RAS messages. In addition to filtering the messages, you can also choose how you view the messages. Figure 2-32 shows the 50 most recent messages in the event log in row format, which is the default view.



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

No Attention Required

0 Fatal
0 Error
0 Info / Warn

20 hours

Filter Options: Most recent 50 events (link)

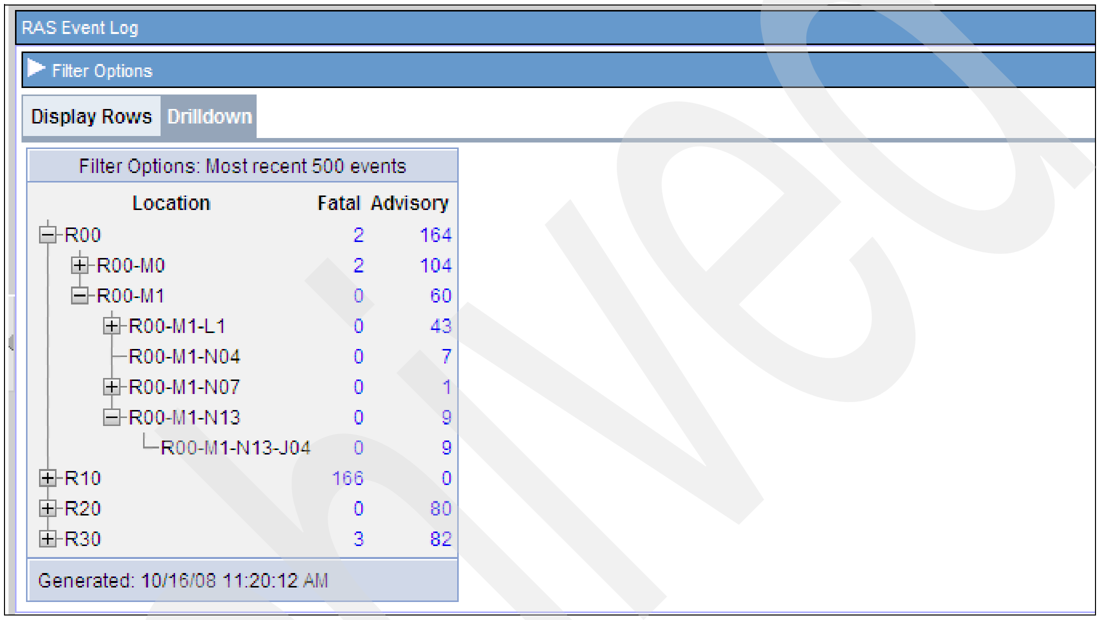
	Event Time	Message ID	Severity	Location	Job ID	Message
i	8/23/07 1:29:50 PM	CARD_0360	Fatal	R00-M0-N11		Problem configuring PTMON facility on a processor card:
i	8/23/07 1:29:50 PM	CARD_053A	Fatal	R00-M0-N11-J21		PTMON sensor did not finish initialization: $(STATUS)
i	8/23/07 1:22:46 PM	CARD_0360	Fatal	R00-M0-N04		Problem configuring PTMON facility on a processor card:
i	8/23/07 1:22:46 PM	CARD_053A	Fatal	R00-M0-N04-J23		PTMON sensor did not finish initialization: $(STATUS)
i	8/23/07 1:22:46 PM	CARD_0360	Fatal	R00-M0-N13		Problem configuring PTMON facility on a processor card:
i	8/23/07 1:22:46 PM	CARD_053A	Fatal	R00-M0-N13-J08		PTMON sensor did not finish initialization: $(STATUS)
i	8/23/07 1:22:44 PM	CARD_0360	Fatal	R00-M0-N10		Problem configuring PTMON facility on a processor card:
i	8/23/07 1:22:44 PM	CARD_053A	Fatal	R00-M0-N10-J09		PTMON sensor did not finish initialization: $(STATUS)
i	8/23/07 1:22:43 PM	CARD_0360	Fatal	R00-M0-N02		Problem configuring PTMON facility on a processor card:
i	8/23/07 1:22:43 PM	CARD_053A	Fatal	R00-M0-N02-J26		PTMON sensor did not finish initialization: $(STATUS)
i	8/23/07 1:22:43 PM	CARD_0360	Fatal	R00-M0-N00		Problem configuring PTMON facility on a processor card:
i	8/23/07 1:22:43 PM	CARD_053A	Fatal	R00-M0-N00-J09		PTMON sensor did not finish initialization: $(STATUS)

Figure 2-32 RAS Event Log

You can select the method that you want to display the messages by clicking the Display Rows or Drilldown tab. Figure 2-33 shows the drill-down method.

You can click the plus (+) symbol to drill down into the hardware to find the location of the error or errors. The numbers in the *Fatal* and *Advisory* columns are links. Clicking a link opens the Display Rows tab with the filter options changed to the corresponding severity and location of the selected link.

For example, clicking the **7** in the Advisory column for R00-M1-N04 returns all the advisory type RAS events (Info and Warn) for that node card. Clicking a link in the Fatal column returns the RAS messages with the severity of Fatal and Error for the chosen location.



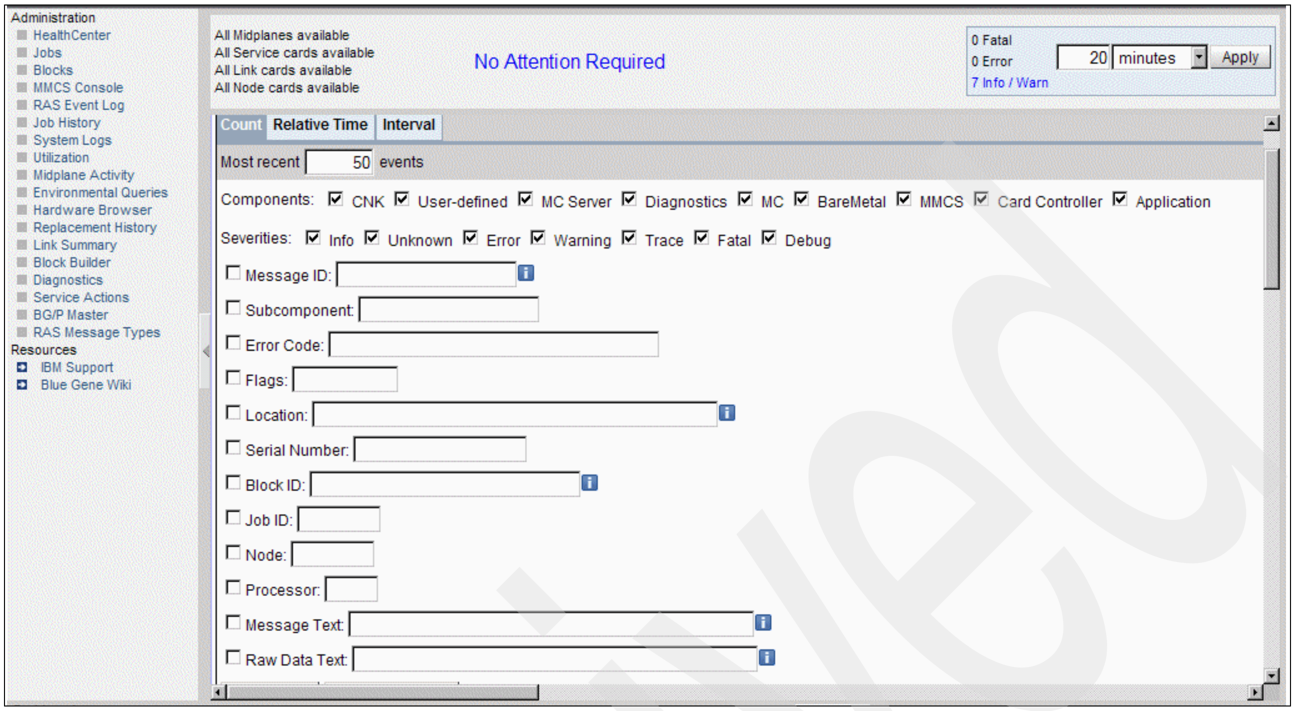
Filter Options: Most recent 500 events

Location	Fatal	Advisory
R00	2	164
R00-M0	2	104
R00-M1	0	60
R00-M1-L1	0	43
R00-M1-N04	0	7
R00-M1-N07	0	1
R00-M1-N13	0	9
R00-M1-N13-J04	0	9
R10	166	0
R20	0	80
R30	3	82

Generated: 10/16/08 11:20:12 AM

Figure 2-33 RAS Event Log Drilldown

Clicking the Filter Option twisty opens the page shown in Figure 2-34.



Administration
- HealthCenter
- Jobs
- Blocks
- MMCS Console
- RAS Event Log
- Job History
- System Logs
- Utilization
- Midplane Activity
- Environmental Queries
- Hardware Browser
- Replacement History
- Link Summary
- Block Builder
- Diagnostics
- Service Actions
- BG/P Master
- RAS Message Types

Resources
- IBM Support
- Blue Gene Wiki

All Midplanes available
All Service cards available
All Link cards available
All Node cards available

No Attention Required

0 Fatal
0 Error
7 Info / Warn

20 minutes Apply

Count | Relative Time | Interval

Most recent 50 events

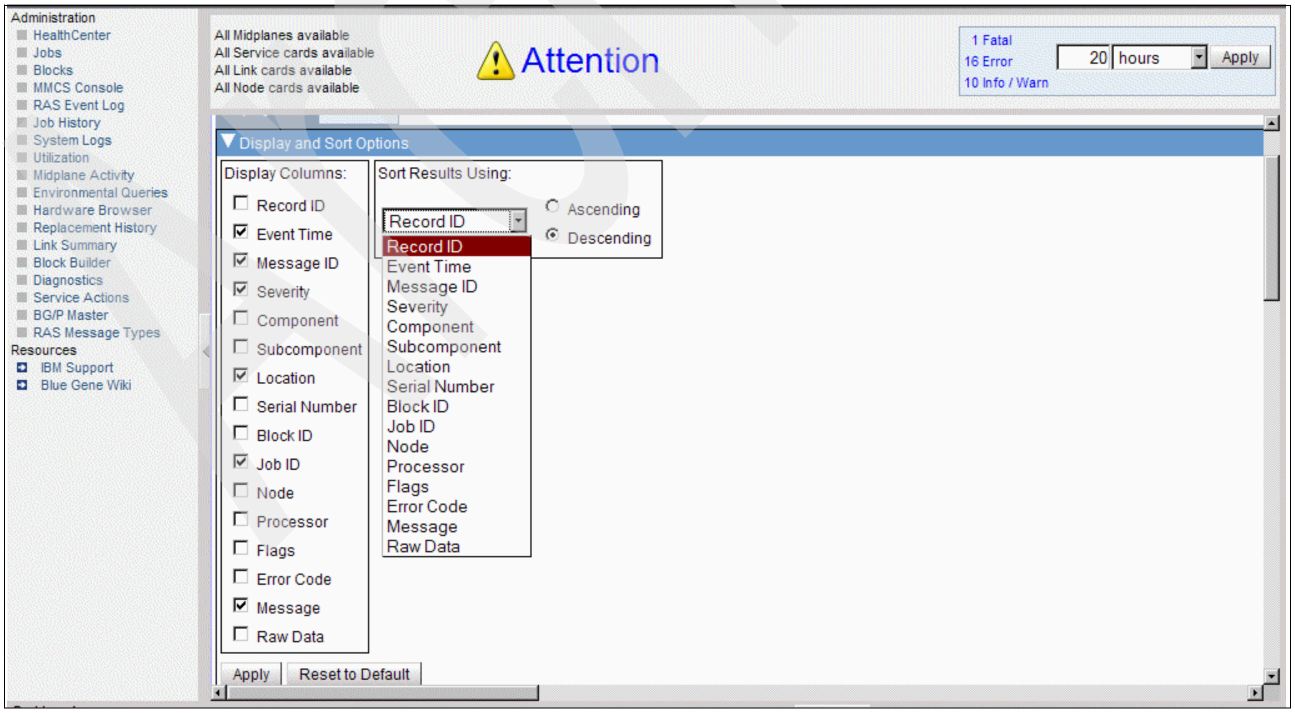
Components: [✓ CNK] [✓ User-defined] [✓ MC Server] [✓ Diagnostics] [✓ MC] [✓ BareMetal] [✓ MMCS] [✓ Card Controller] [✓ Application]

Severities: [✓ Info] [✓ Unknown] [✓ Error] [✓ Warning] [✓ Trace] [✓ Fatal] [✓ Debug]

- Message ID:
- Subcomponent:
- Error Code:
- Flags:
- Location:
- Serial Number:
- Block ID:
- Job ID:
- Node:
- Processor:
- Message Text:
- Raw Data Text:

Figure 2-34 RAS Event Log Filter Options

Just below the Filter Options are the Display and Sort Options. Expanding this section opens the page shown in Figure 2-35. By selecting or deselecting the various column heads you can customize the display. The drop-down menu allows you to select the column head that you sort by. Selecting either radio button sorts the results in ascending or descending order.



Administration
- HealthCenter
- Jobs
- Blocks
- MMCS Console
- RAS Event Log
- Job History
- System Logs
- Utilization
- Midplane Activity
- Environmental Queries
- Hardware Browser
- Replacement History
- Link Summary
- Block Builder
- Diagnostics
- Service Actions
- BG/P Master
- RAS Message Types

Resources
- IBM Support
- Blue Gene Wiki

All Midplanes available
All Service cards available
All Link cards available
All Node cards available

Attention

1 Fatal
16 Error
10 Info / Warn

20 hours Apply

Display and Sort Options

Display Columns:
- Record ID
- [✓ Event Time]
- [✓ Message ID]
- [✓ Severity]
- Component
- Subcomponent
- [✓ Location]
- Serial Number
- Block ID
- [✓ Job ID]
- Node
- Processor
- Flags
- Error Code
- [✓ Message]
- Raw Data

Sort Results Using:

Record ID

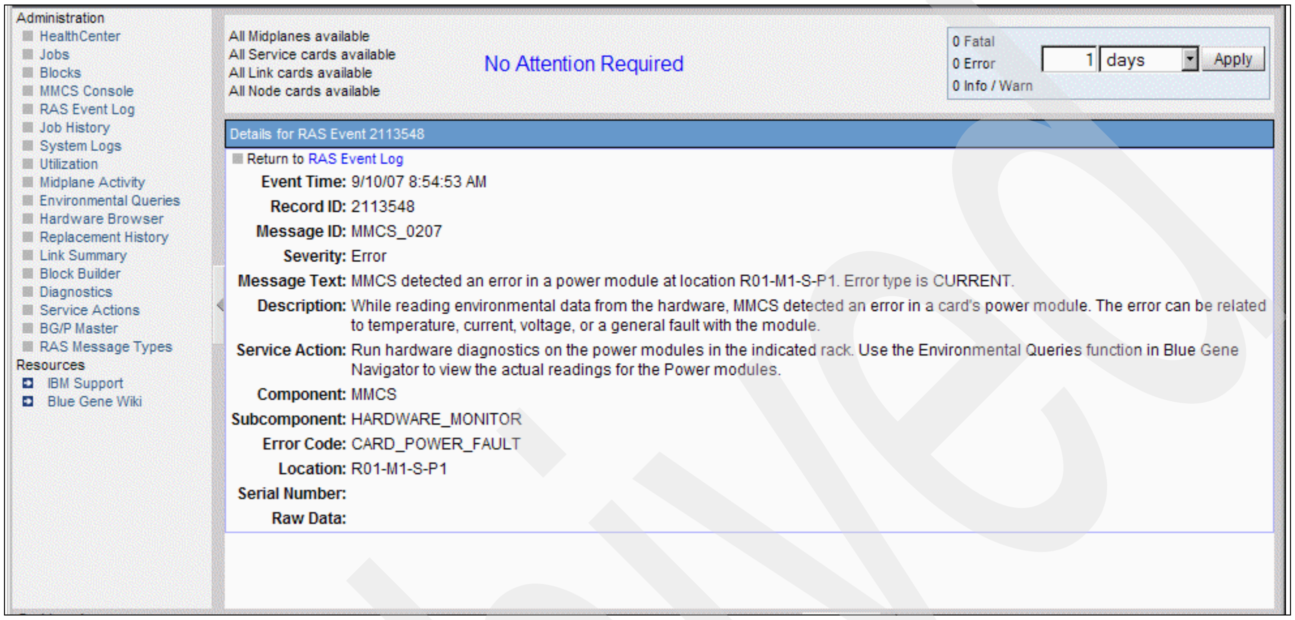
Record ID
Event Time
Message ID
Severity
Component
Subcomponent
Location
Serial Number
Block ID
Job ID
Node
Processor
Flags
Error Code
Message
Raw Data

Ascending [✓ Descending]

Apply | Reset to Default

Figure 2-35 RAS event sort options

Notice the additional information icon in the first column (extreme left side) of the content area table in Figure 2-32 on page 49. Clicking that icon opens a RAS event detail page similar to the one shown in Figure 2-36. The details include the date and time that the error occurred, as well as a record ID, message ID, and the severity of the event. Beyond the general information, the page contains information that relates to the specific event, including a description of the error and information about how to further handle the problem.



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

No Attention Required

Details for RAS Event 2113548

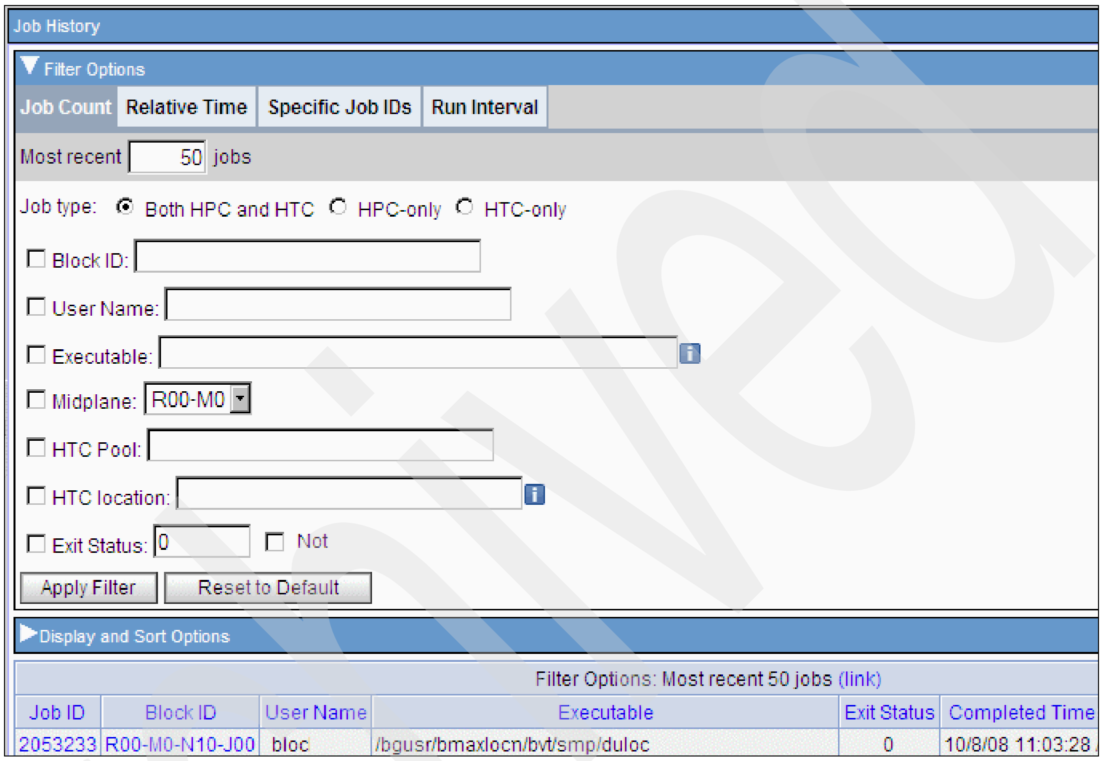
Return to RAS Event Log

Event Time:	9/10/07 8:54:53 AM
Record ID:	2113548
Message ID:	MMCS_0207
Severity:	Error
Message Text:	MMCS detected an error in a power module at location R01-M1-S-P1. Error type is CURRENT.
Description:	While reading environmental data from the hardware, MMCS detected an error in a card's power module. The error can be related to temperature, current, voltage, or a general fault with the module.
Service Action:	Run hardware diagnostics on the power modules in the indicated rack. Use the Environmental Queries function in Blue Gene Navigator to view the actual readings for the Power modules.
Component:	MMCS
Subcomponent:	HARDWARE_MONITOR
Error Code:	CARD_POWER_FAULT
Location:	R01-M1-S-P1
Serial Number:	
Raw Data:	

Figure 2-36 RAS event detail page

2.3.6 Job History

The Job History page, shown in Figure 2-37, by default shows the 50 most recent jobs that have run on the system. You can filter the data shown on the page by job type, specific user, block, or midplane. You can also query on the exit status, executable, or a single job identification number. The *HTC Pool* filter provides a means of narrowing the results displayed based on jobs that ran in a particular pool. The *HTC location* filter allows you to narrow the search results to a specific compute node or core.


Job History

Filter Options

Job Count | Relative Time | Specific Job IDs | Run Interval

Most recent: 50 jobs

Job type: [✓ Both HPC and HTC] HPC-only HTC-only

☐ Block ID:
☐ User Name:
☐ Executable:
☐ Midplane: R00-M0
☐ HTC Pool:
☐ HTC location:
☐ Exit Status: 0 ☐ Not

Apply Filter | Reset to Default

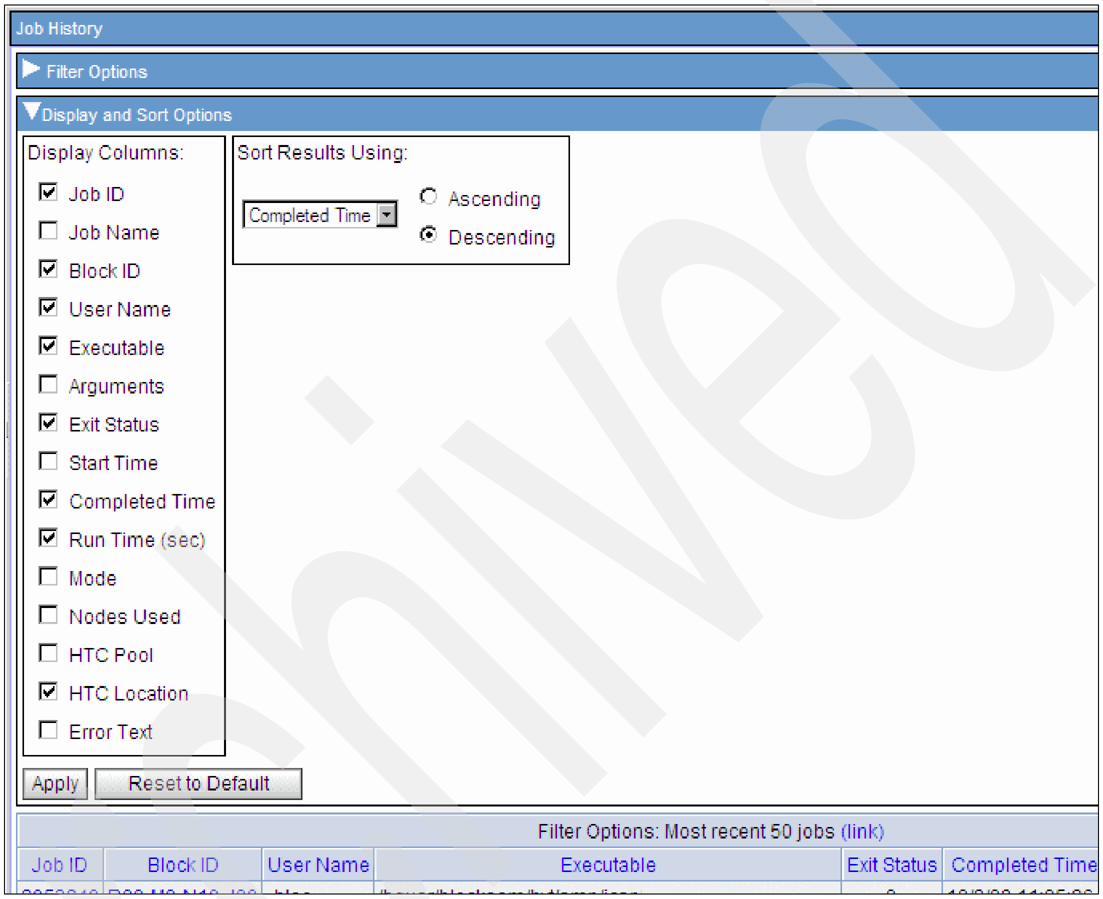
Display and Sort Options

Filter Options: Most recent 50 jobs (link)

Job ID	Block ID	User Name	Executable	Exit Status	Completed Time
2053233	R00-M0-N10-J00	bloc	/bgusr/bmaxlocn/bvt/smp/duloc	0	10/8/08 11:03:28

Figure 2-37 Job History

There are several fields in the Job History table that are not displayed by default. Starting in V1R3M0, you can configure the fields that are displayed in the results using the Display and Sort Options. By selecting the check boxes next to the columns that you want displayed, you can customize the view of the job history table contents (Figure 2-38). Additionally, you can sort in ascending or descending order by any of the columns. Note that the "(link)" in the results table header encodes the configured display and sort options. You can copy and paste this link to another location to re-create the results.



Display Columns:

- [✓] Job ID
- [ ] Job Name
- [✓] Block ID
- [✓] User Name
- [✓] Executable
- [ ] Arguments
- [✓] Exit Status
- [ ] Start Time
- [✓] Completed Time
- [✓] Run Time (sec)
- [ ] Mode
- [ ] Nodes Used
- [ ] HTC Pool
- [✓] HTC Location
- [ ] Error Text

Sort Results Using: Completed Time — Ascending [✓ Descending]

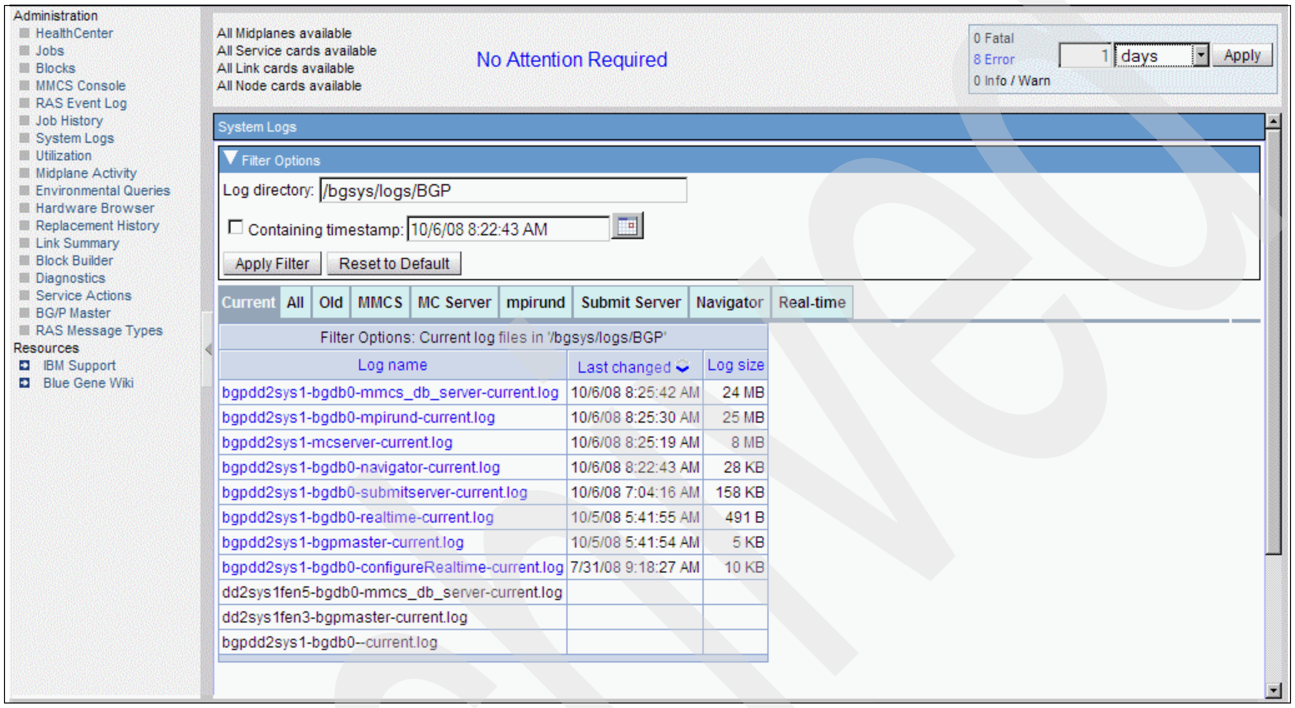
Filter Options: Most recent 50 jobs (link)

Job ID	Block ID	User Name	Executable	Exit Status	Completed Time

Figure 2-38 Display and Sort Options

2.3.7 System Logs

Clicking the System Logs link in the navigation pane opens the page shown in Figure 2-39. From this page, you can view all the logs on the system—the historical logs (old) or the logs that are presently being written to (current). By default, system logs are created in the /bgsys/logs/BGP directory. To view logs in other directories, supply the path in the Filter Options *Log directory* text box, and click **Apply Filter** to display the contents of the selected directory that have a `.log` file extension. You can also filter the logs that display by selecting a date and time stamp and then by clicking **Apply Filter**.



Log directory: /bgsys/logs/BGP

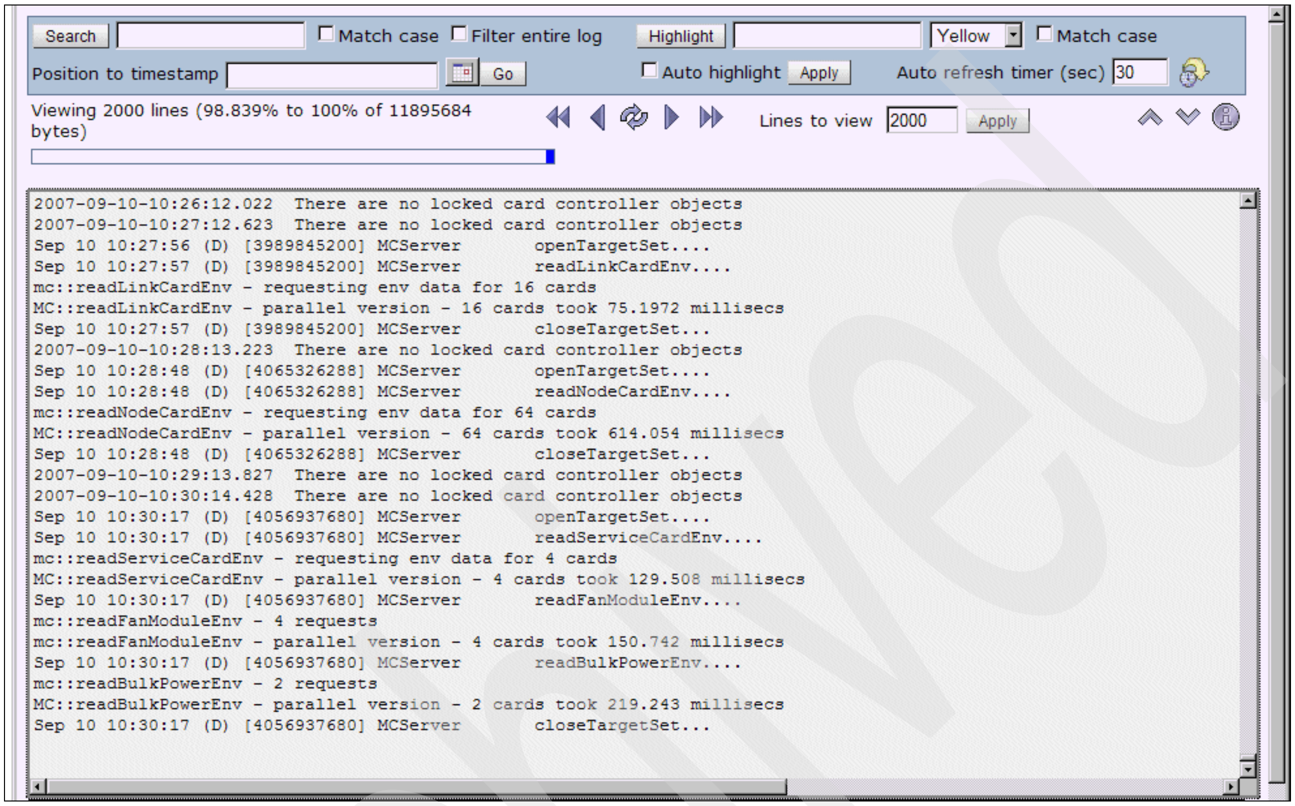
Containing timestamp: 10/6/08 8:22:43 AM

Filter Options: Current log files in '/bgsys/logs/BGP'

Log name	Last changed	Log size
bgpdd2sys1-bgdb0-mmcs_db_server-current.log	10/6/08 8:25:42 AM	24 MB
bgpdd2sys1-bgdb0-mpirund-current.log	10/6/08 8:25:30 AM	25 MB
bgpdd2sys1-mcserver-current.log	10/6/08 8:25:19 AM	8 MB
bgpdd2sys1-bgdb0-navigator-current.log	10/6/08 8:22:43 AM	28 KB
bgpdd2sys1-bgdb0-submitserver-current.log	10/6/08 7:04:16 AM	158 KB
bgpdd2sys1-bgdb0-realtime-current.log	10/5/08 5:41:55 AM	491 B
bgpdd2sys1-bgpmaster-current.log	10/5/08 5:41:54 AM	5 KB
bgpdd2sys1-bgdb0-configureRealtime-current.log	7/31/08 9:18:27 AM	10 KB
dd2sys1fen5-bgdb0-mmcs_db_server-current.log		
dd2sys1fen3-bgpmaster-current.log		
bgpdd2sys1-bgdb0--current.log		

Figure 2-39 System Logs

Each of the file names in the Log name column of the System Logs page is a link to the file itself. Clicking one of the file names opens a window similar to the one shown in Figure 2-40. The Log Viewer feature of the Navigator allows you to view both current and historical logs and to perform a variety of searches within the log.



```
2007-09-10-10:26:12.022  There are no locked card controller objects
2007-09-10-10:27:12.623  There are no locked card controller objects
Sep 10 10:27:56 (D) [3989845200] MCServer      openTargetSet....
Sep 10 10:27:57 (D) [3989845200] MCServer      readLinkCardEnv....
mc::readLinkCardEnv - requesting env data for 16 cards
MC::readLinkCardEnv - parallel version - 16 cards took 75.1972 millisecs
Sep 10 10:27:57 (D) [3989845200] MCServer      closeTargetSet...
2007-09-10-10:28:13.223  There are no locked card controller objects
Sep 10 10:28:48 (D) [4065326288] MCServer      openTargetSet....
Sep 10 10:28:48 (D) [4065326288] MCServer      readNodeCardEnv....
mc::readNodeCardEnv - requesting env data for 64 cards
MC::readNodeCardEnv - parallel version - 64 cards took 614.054 millisecs
Sep 10 10:28:48 (D) [4065326288] MCServer      closeTargetSet...
2007-09-10-10:29:13.827  There are no locked card controller objects
2007-09-10-10:30:14.428  There are no locked card controller objects
Sep 10 10:30:17 (D) [4056937680] MCServer      openTargetSet....
Sep 10 10:30:17 (D) [4056937680] MCServer      readServiceCardEnv....
mc::readServiceCardEnv - requesting env data for 4 cards
MC::readServiceCardEnv - parallel version - 4 cards took 129.508 millisecs
Sep 10 10:30:17 (D) [4056937680] MCServer      readFanModuleEnv....
mc::readFanModuleEnv - 4 requests
mc::readFanModuleEnv - parallel version - 4 cards took 150.742 millisecs
Sep 10 10:30:17 (D) [4056937680] MCServer      readBulkPowerEnv....
mc::readBulkPowerEnv - 2 requests
MC::readBulkPowerEnv - parallel version - 2 cards took 219.243 millisecs
Sep 10 10:30:17 (D) [4056937680] MCServer      closeTargetSet...
```

Figure 2-40 Log viewer window

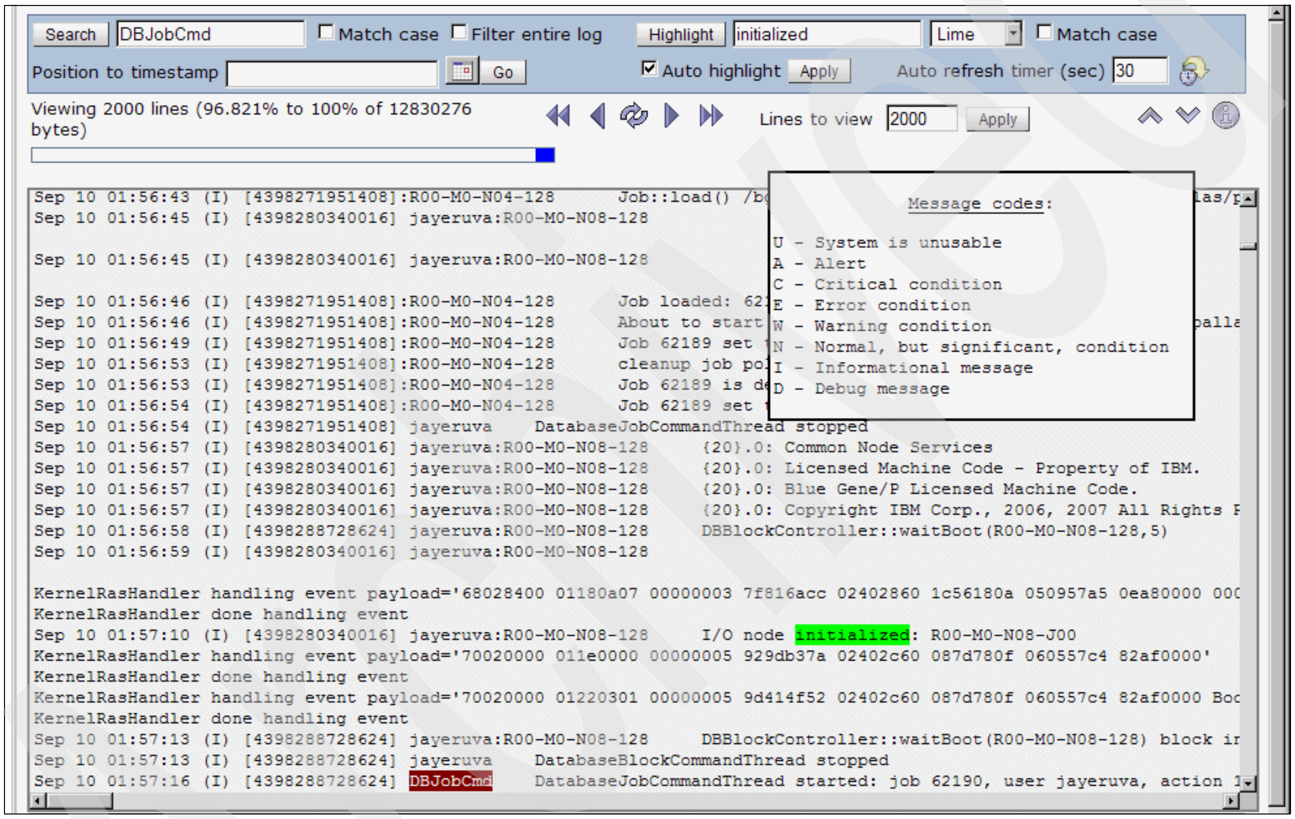
Figure 2-41 shows a number of log viewer features. These features assist you in the following areas:

- *Movement*: You can move around in the log file and visualize where you are in the log file. Movement can be by paging backwards and forwards or by moving to a specific time stamp or other search string.
- *Refresh*: When viewing a current (live) log file, it might be growing in size or rolling over to become a new log file, while you are viewing it. You can refresh your log file view manually or automatically.
- *Filtering*: You can select only the log file lines of interest, in a manner similar to the Linux **grep** command. This filtering can be applied to the entire log file or to the lines that are currently being viewed.
- *Highlighting*: Two independent highlighting schemes are available. In Figure 2-41, the auto highlight feature displays lines in various colors (for example, error messages are displayed in red), and the manual highlight feature is highlighting the text *job* in line.
- *Hints*: Hover your mouse over any of the log file viewer icons, and the viewer provides a *hint* that describes that icon's function. In addition, if you hover your mouse over the information (*i*) icon, the log viewer displays the message code descriptions as shown in Figure 2-41.

You can copy and paste from the log view to any of the form fields. For example, you can search for a specific message, copy that message's times tamp text into the *position to timestamp* box, and then click **Go** to display the full log beginning at that time stamp.

The search box provides two different modes of searching, depending on whether the *filter entire log* box is selected. If the box is *not* selected, clicking **Search** causes the browser to search the lines that are currently being viewed. This function is identical to using the browser Find function, except that the log viewer limits the search to the log file lines.

The log viewer does an entirely different type of search when you select the *Filter entire log* box. In this case, the log viewer does a server-side search, returning only those lines that match the search string. All search strings are fixed strings. The log viewer does not recognize wildcard characters or regular expressions.



Search: DBJobCmd ☐ Match case ☐ Filter entire log

Highlight: initialized, Lime ☐ Match case

Position to timestamp:

☑ Auto highlight

Auto refresh timer (sec) 30

Viewing 2000 lines (96.821% to 100% of 12830276 bytes)

Lines to view 2000

```
Sep 10 01:56:43 (I) [4398271951408]:R00-M0-N04-128      Job::load() /bg                                       las/p
Sep 10 01:56:45 (I) [4398280340016] jayeruva:R00-M0-N08-128

Sep 10 01:56:45 (I) [4398280340016] jayeruva:R00-M0-N08-128

Sep 10 01:56:46 (I) [4398271951408]:R00-M0-N04-128      Job loaded: 621
Sep 10 01:56:46 (I) [4398271951408]:R00-M0-N04-128      About to start                                        palla
Sep 10 01:56:49 (I) [4398271951408]:R00-M0-N04-128      Job 62189 set t
Sep 10 01:56:53 (I) [4398271951408]:R00-M0-N04-128      cleanup job pol
Sep 10 01:56:53 (I) [4398271951408]:R00-M0-N04-128      Job 62189 is de
Sep 10 01:56:54 (I) [4398271951408]:R00-M0-N04-128      Job 62189 set t
Sep 10 01:56:54 (I) [4398271951408] jayeruva    DatabaseJobCommandThread stopped
Sep 10 01:56:57 (I) [4398280340016] jayeruva:R00-M0-N08-128     {20}.0: Common Node Services
Sep 10 01:56:57 (I) [4398280340016] jayeruva:R00-M0-N08-128     {20}.0: Licensed Machine Code - Property of IBM.
Sep 10 01:56:57 (I) [4398280340016] jayeruva:R00-M0-N08-128     {20}.0: Blue Gene/P Licensed Machine Code.
Sep 10 01:56:57 (I) [4398280340016] jayeruva:R00-M0-N08-128     {20}.0: Copyright IBM Corp., 2006, 2007 All Rights F
Sep 10 01:56:58 (I) [4398288728624] jayeruva:R00-M0-N08-128     DBBlockController::waitBoot(R00-M0-N08-128,5)
Sep 10 01:56:59 (I) [4398280340016] jayeruva:R00-M0-N08-128

KernelRasHandler handling event payload='68028400 01180a07 00000003 7f816acc 02402860 1c56180a 050957a5 0ea80000 00C
KernelRasHandler done handling event
Sep 10 01:57:10 (I) [4398280340016] jayeruva:R00-M0-N08-128     I/O node initialized: R00-M0-N08-J00
KernelRasHandler handling event payload='70020000 011e0000 00000005 929db37a 02402c60 087d780f 060557c4 82af0000'
KernelRasHandler done handling event
KernelRasHandler handling event payload='70020000 01220301 00000005 9d414f52 02402c60 087d780f 060557c4 82af0000 Boc
KernelRasHandler done handling event
Sep 10 01:57:13 (I) [4398288728624] jayeruva:R00-M0-N08-128     DBBlockController::waitBoot(R00-M0-N08-128) block ir
Sep 10 01:57:13 (I) [4398288728624] jayeruva    DatabaseBlockCommandThread stopped
Sep 10 01:57:16 (I) [4398288728624] DBJobCmd    DatabaseJobCommandThread started: job 62190, user jayeruva, action 1
```

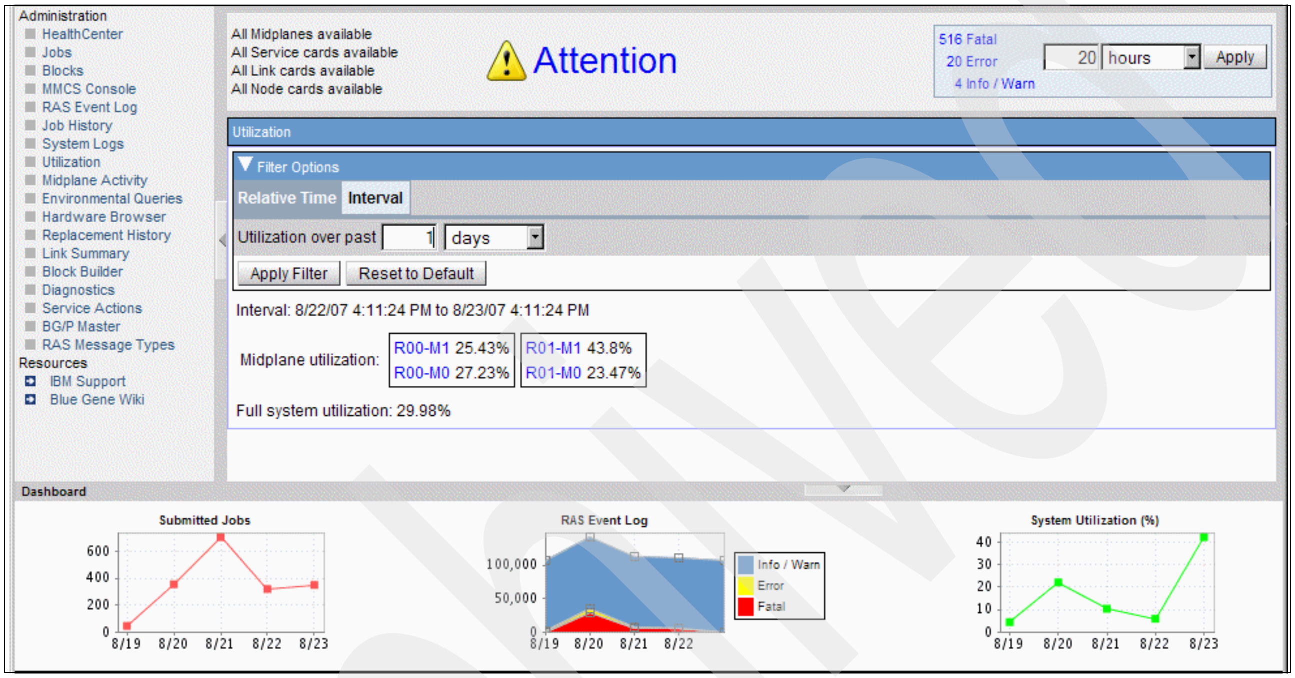
```
Message codes:

U - System is unusable
A - Alert
C - Critical condition
E - Error condition
W - Warning condition
N - Normal, but significant, condition
I - Informational message
D - Debug message
```

Figure 2-41 Log viewer features

2.3.8 Utilization

Figure 2-42 shows the page that displays when you click **Utilization**. This page reports the system utilization, by midplane, for the time specified in the filter options. The Filter Options on this page allow you to select the amount of time that is reflected in the report. The options include *Relative Time* and *Interval*. Relative Time allows you to select a period of time, beginning at the present time, at which you want to go return in history. You supply the numeric value, and the drop-down menu gives you a choice of seconds, minutes, hours, and days.



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

Utilization over past 1 days

Interval: 8/22/07 4:11:24 PM to 8/23/07 4:11:24 PM

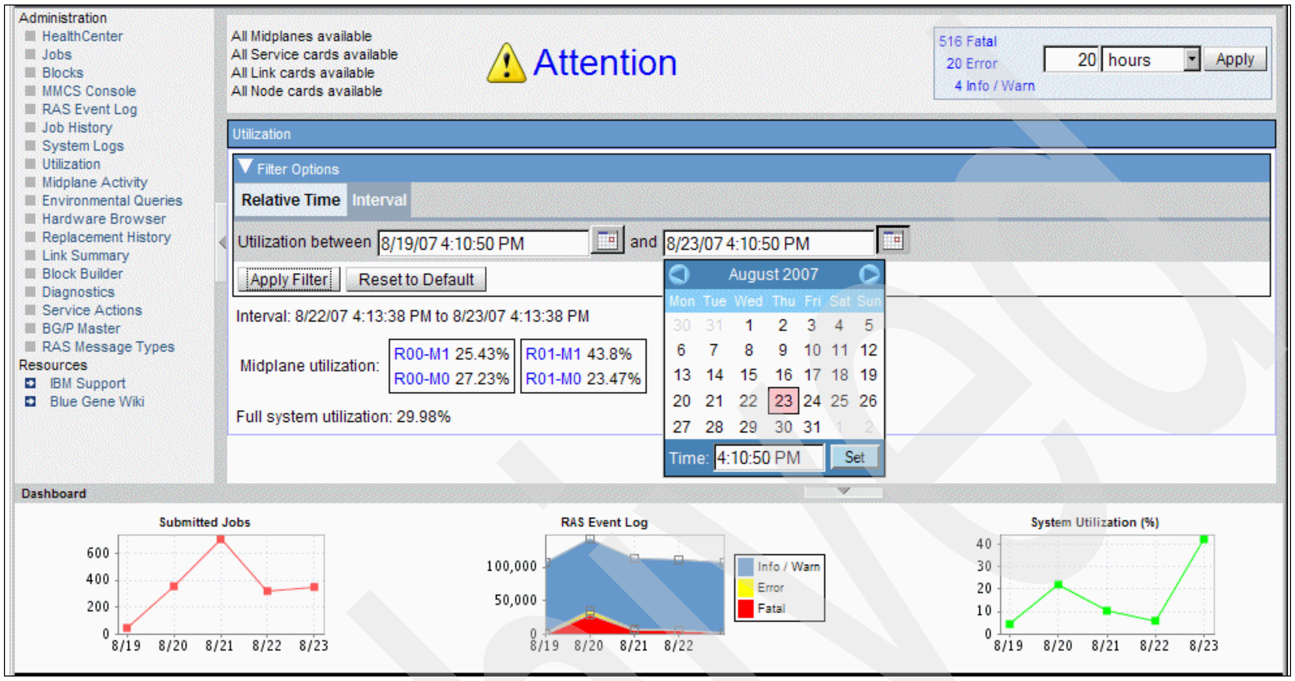
Midplane utilization: R00-M1 25.43% R01-M1 43.8% R00-M0 27.23% R01-M0 23.47%

Full system utilization: 29.98%

Figure 2-42 Utilization page

The Interval option allows you to pick a specific period of time on which to base the results. You can enter the dates manually using the format `dd/mm/yy h:mm:ss AM/PM`. Alternatively, you can click the calendar icon and select the date from the calendar drop-down menu that displays. The default time period for both options is 1 day.

Regardless of the option that you choose, the results are returned by midplane. The location of the midplane is followed by the percentage of utilization for the time period that you chose. The midplane location is actually a link. Clicking the location link queries the database and returns a job history page that contains all of the jobs that ran during the specified time period (refer to Figure 2-43).



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

516 Fatal
20 Error
4 Info / Warn

20 hours

Utilization between 8/19/07 4:10:50 PM and 8/23/07 4:10:50 PM

Time: 4:10:50 PM

Interval: 8/22/07 4:13:38 PM to 8/23/07 4:13:38 PM

Midplane utilization:	R00-M1 25.43%	R01-M1 43.8%
	R00-M0 27.23%	R01-M0 23.47%

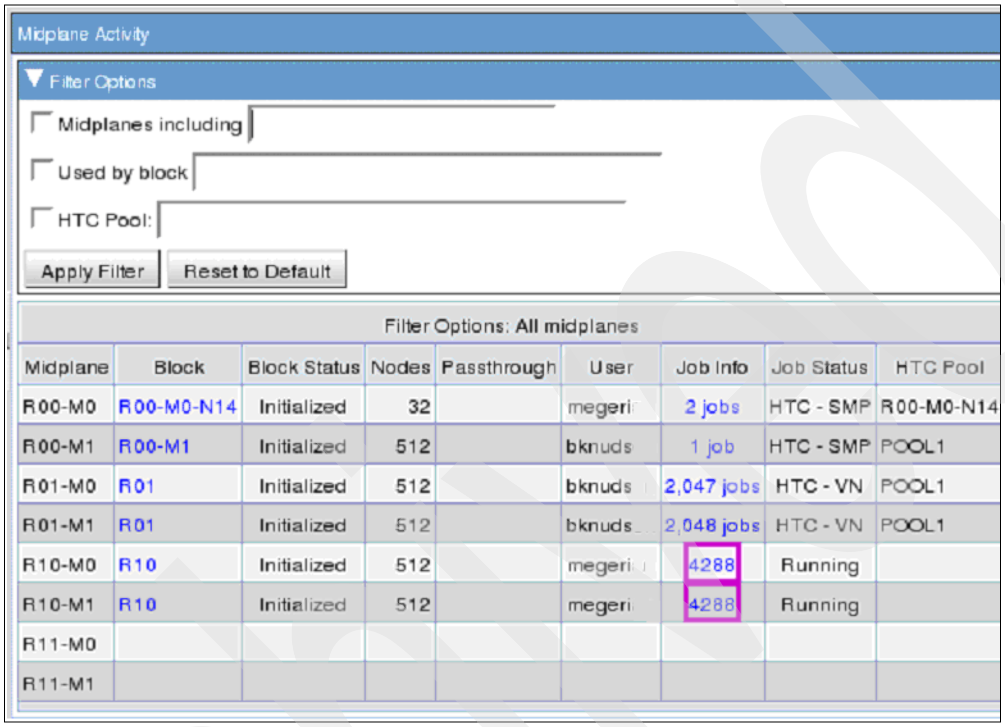
Full system utilization: 29.98%

Figure 2-43 Interval option

2.3.9 Midplane Activity

The Midplane Activity page shows all the current activity on the system. Clicking the link opens the page shown in Figure 2-44. You can filter the output to include the entire system (all midplanes), a single midplane, a selected block, or an HTC pool. The tool reports any blocks in use on the hardware, as well as the status and the current user of that block. You can also determine whether the midplane's link cards are being used to pass through to other midplanes to complete a torus. Any jobs that are currently running on the midplane also display. Both the *Block* and *Job* columns contain links to their respective summary pages.

Figure 2-44 shows that the Filter Options for Midplane activity include a filter for HTC Pools. Looking further down the page, block R01 (spans midplanes R01-M0 and R01-M1) boots in virtual node mode for HTC in pool POOL1. HTC is running 4095 jobs on this rack, with 2047 of them on R01-M0, and 2,048 on R01-M1. It also shows that the pool, POOL1 includes another midplane booted in SMP mode. Clicking the job count for the midplane shows the Job Summary page with the filter set to display those jobs.



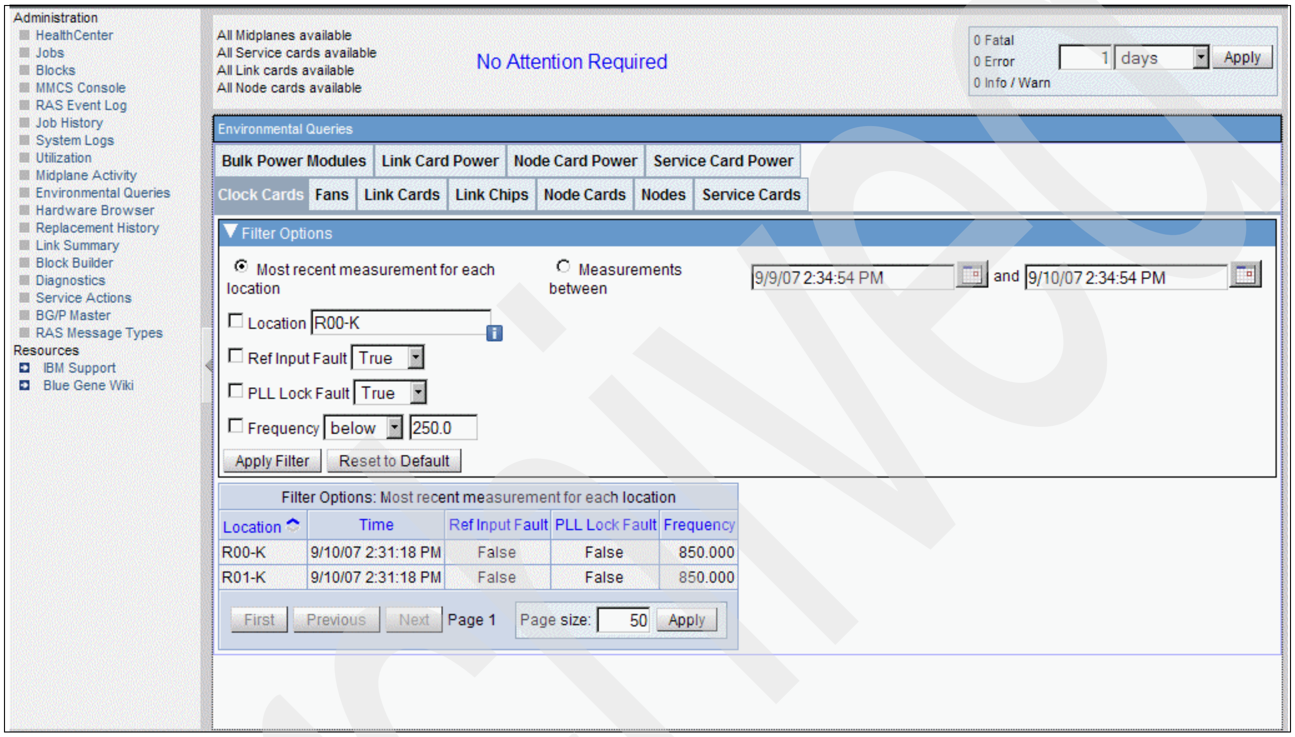
Filter Options: All midplanes

Midplane	Block	Block Status	Nodes	Passthrough	User	Job Info	Job Status	HTC Pool
R00-M0	R00-M0-N14	Initialized	32		megeri	2 jobs	HTC - SMP	R00-M0-N14
R00-M1	R00-M1	Initialized	512		bknuds	1 job	HTC - SMP	POOL1
R01-M0	R01	Initialized	512		bknuds	2,047 jobs	HTC - VN	POOL1
R01-M1	R01	Initialized	512		bknuds	2,048 jobs	HTC - VN	POOL1
R10-M0	R10	Initialized	512		megeri	4288	Running	
R10-M1	R10	Initialized	512		megeri	4288	Running	
R11-M0								
R11-M1								

Figure 2-44 Midplane Activity

2.3.10 Environmental Queries

The Environmental Queries section of the Navigator provides access to historical data collected by the hardware monitoring functions in the MMCS server. Figure 2-45 shows the main page and filter options that are available when you click the Environmental Options link in the navigation pane. Clicking any of the tabs opens a new page that shows the most current readings taken by the Hardware Monitor. Expanding the Filter Options on each of the pages shows the options that are available relative to the hardware tab that is selected. For more information about the Environmental Monitor, see 5.7, "Environmental Monitor" on page 129.


Administration
- HealthCenter
- Jobs
- Blocks
- MMCS Console
- RAS Event Log
- Job History
- System Logs
- Utilization
- Midplane Activity
- Environmental Queries
- Hardware Browser
- Replacement History
- Link Summary
- Block Builder
- Diagnostics
- Service Actions
- BG/P Master
- RAS Message Types

Resources
- IBM Support
- Blue Gene Wiki

All Midplanes available
All Service cards available
All Link cards available
All Node cards available

No Attention Required

0 Fatal
0 Error
0 Info / Warn
1 days Apply

Environmental Queries

Bulk Power Modules | Link Card Power | Node Card Power | Service Card Power
Clock Cards | Fans | Link Cards | Link Chips | Node Cards | Nodes | Service Cards

Filter Options

[✓ Most recent measurement for each location] Measurements between 9/9/07 2:34:54 PM and 9/10/07 2:34:54 PM

Location R00-K
Ref Input Fault True
PLL Lock Fault True
Frequency below 250.0

Apply Filter Reset to Default

Filter Options: Most recent measurement for each location

Location	Time	Ref Input Fault	PLL Lock Fault	Frequency
R00-K	9/10/07 2:31:18 PM	False	False	850.000
R01-K	9/10/07 2:31:18 PM	False	False	850.000

First Previous Next Page 1 Page size: 50 Apply

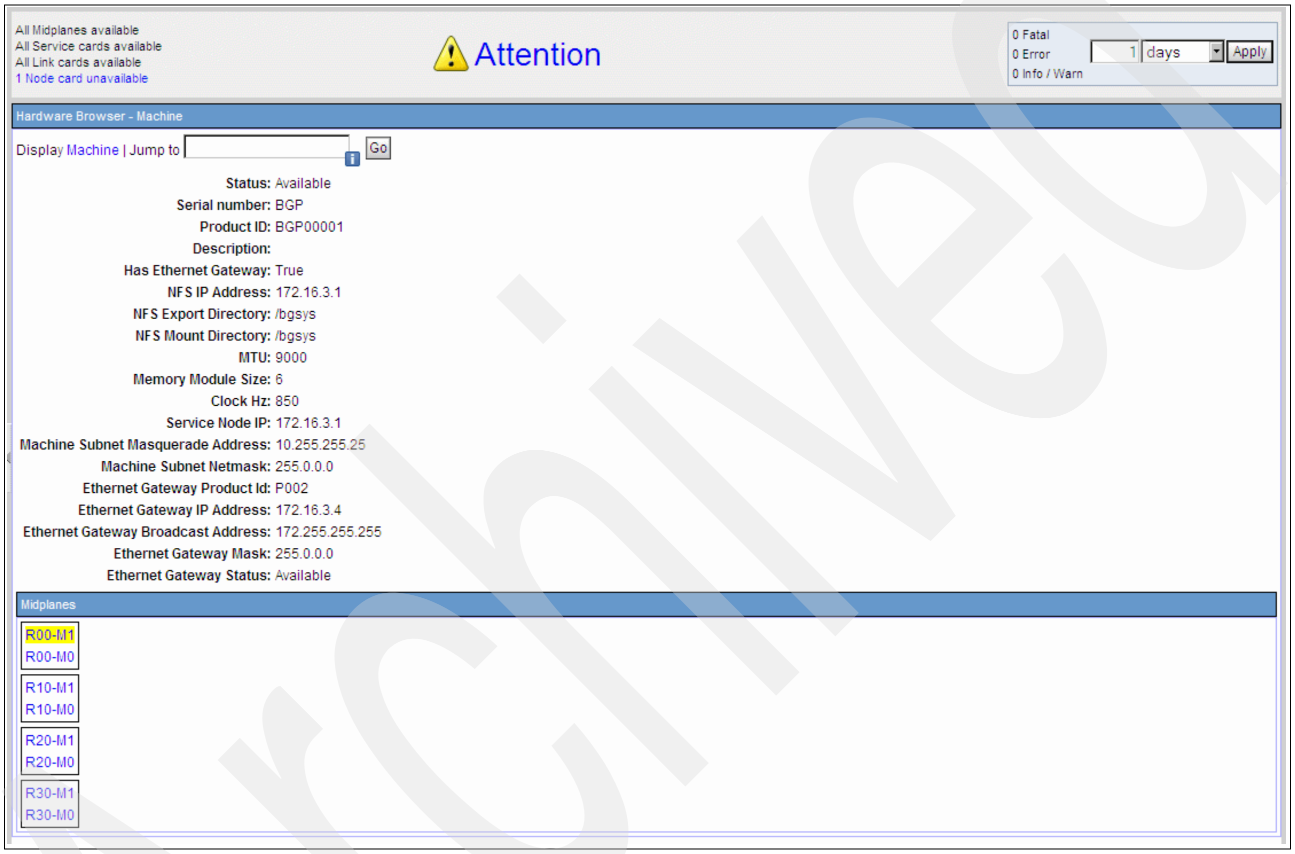
Figure 2-45 Environmental Queries

2.3.11 Hardware Browser

The initial Hardware Browser page contains information about the system's configuration (refer to Figure 2-46). This information is a combination of several of the DB2 tables:

- BGPMACHINE
- BGPETHGATEWAY
- BGPMACHINESUBNET

Throughout the Hardware Browser section of Navigator, hardware that is unavailable is highlighted in yellow. For example, in Figure 2-46, all or part of R00-M1 is unavailable.

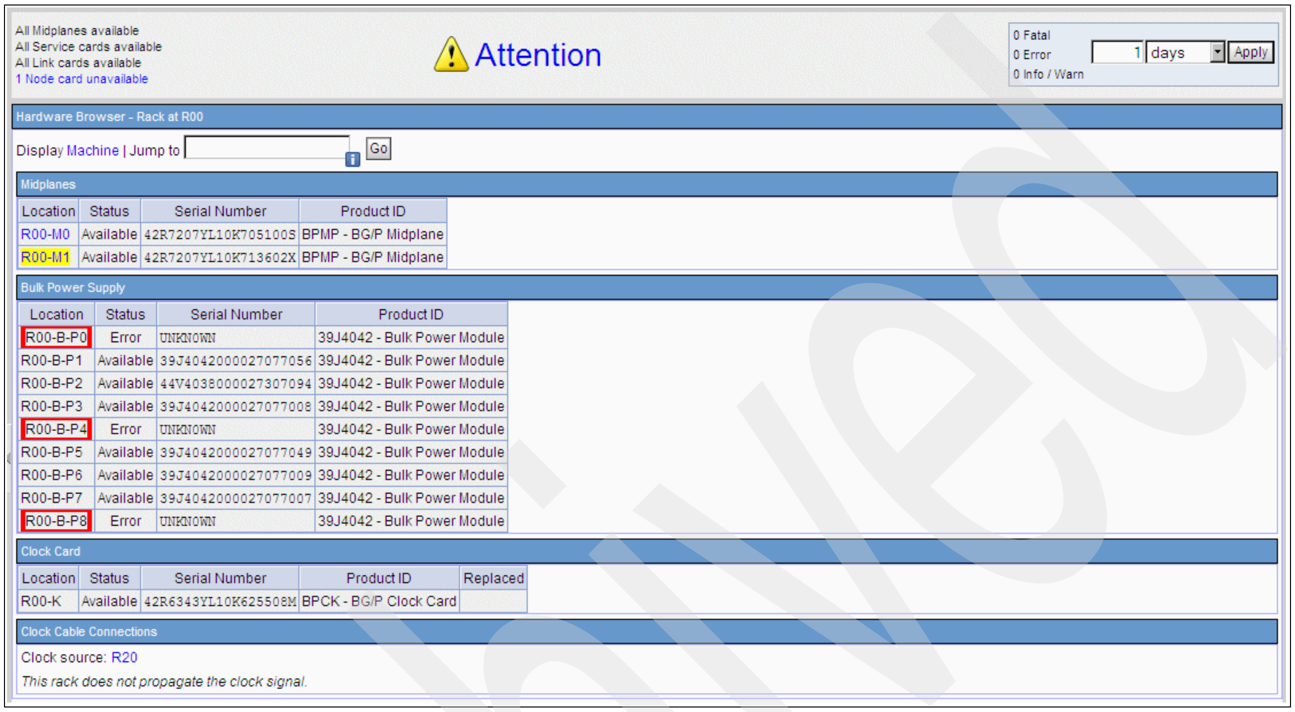


All Midplanes available
All Service cards available
All Link cards available
1 Node card unavailable

Status:	Available
Serial number:	BGP
Product ID:	BGP00001
Description:	
Has Ethernet Gateway:	True
NFS IP Address:	172.16.3.1
NFS Export Directory:	/bgsys
NFS Mount Directory:	/bgsys
MTU:	9000
Memory Module Size:	6
Clock Hz:	850
Service Node IP:	172.16.3.1
Machine Subnet Masquerade Address:	10.255.255.25
Machine Subnet Netmask:	255.0.0.0
Ethernet Gateway Product Id:	P002
Ethernet Gateway IP Address:	172.16.3.4
Ethernet Gateway Broadcast Address:	172.255.255.255
Ethernet Gateway Mask:	255.0.0.0
Ethernet Gateway Status:	Available

Figure 2-46 Hardware Browser link

You can click any of the links at the bottom of the page, and information that pertains to the specified rack displays. For example, we clicked the highlighted link, R00-M1, to open the page shown in Figure 2-47. Any hardware that is not available is flagged with the appropriate status. Note that R00-M1 is still highlighted. This page contains information about each of the midplanes, Bulk Power Modules (BPMs), Clock card, and clock signal distribution.



All Midplanes available
All Service cards available
All Link cards available
1 Node card unavailable

0 Fatal
0 Error
0 Info / Warn
1 days

Hardware Browser - Rack at R00

Display Machine | Jump to

Midplanes

Location	Status	Serial Number	Product ID
R00-M0	Available	42R7207YL10K705100S	BPMP - BG/P Midplane
R00-M1	Available	42R7207YL10K713602X	BPMP - BG/P Midplane

Bulk Power Supply

Location	Status	Serial Number	Product ID
R00-B-P0	Error	UNKNOWN	39J4042 - Bulk Power Module
R00-B-P1	Available	39J4042000027077056	39J4042 - Bulk Power Module
R00-B-P2	Available	44V4038000027307094	39J4042 - Bulk Power Module
R00-B-P3	Available	39J4042000027077008	39J4042 - Bulk Power Module
R00-B-P4	Error	UNKNOWN	39J4042 - Bulk Power Module
R00-B-P5	Available	39J4042000027077049	39J4042 - Bulk Power Module
R00-B-P6	Available	39J4042000027077009	39J4042 - Bulk Power Module
R00-B-P7	Available	39J4042000027077007	39J4042 - Bulk Power Module
R00-B-P8	Error	UNKNOWN	39J4042 - Bulk Power Module

Clock Card

Location	Status	Serial Number	Product ID	Replaced
R00-K	Available	42R6343YL10K625508M	BPCK - BG/P Clock Card	

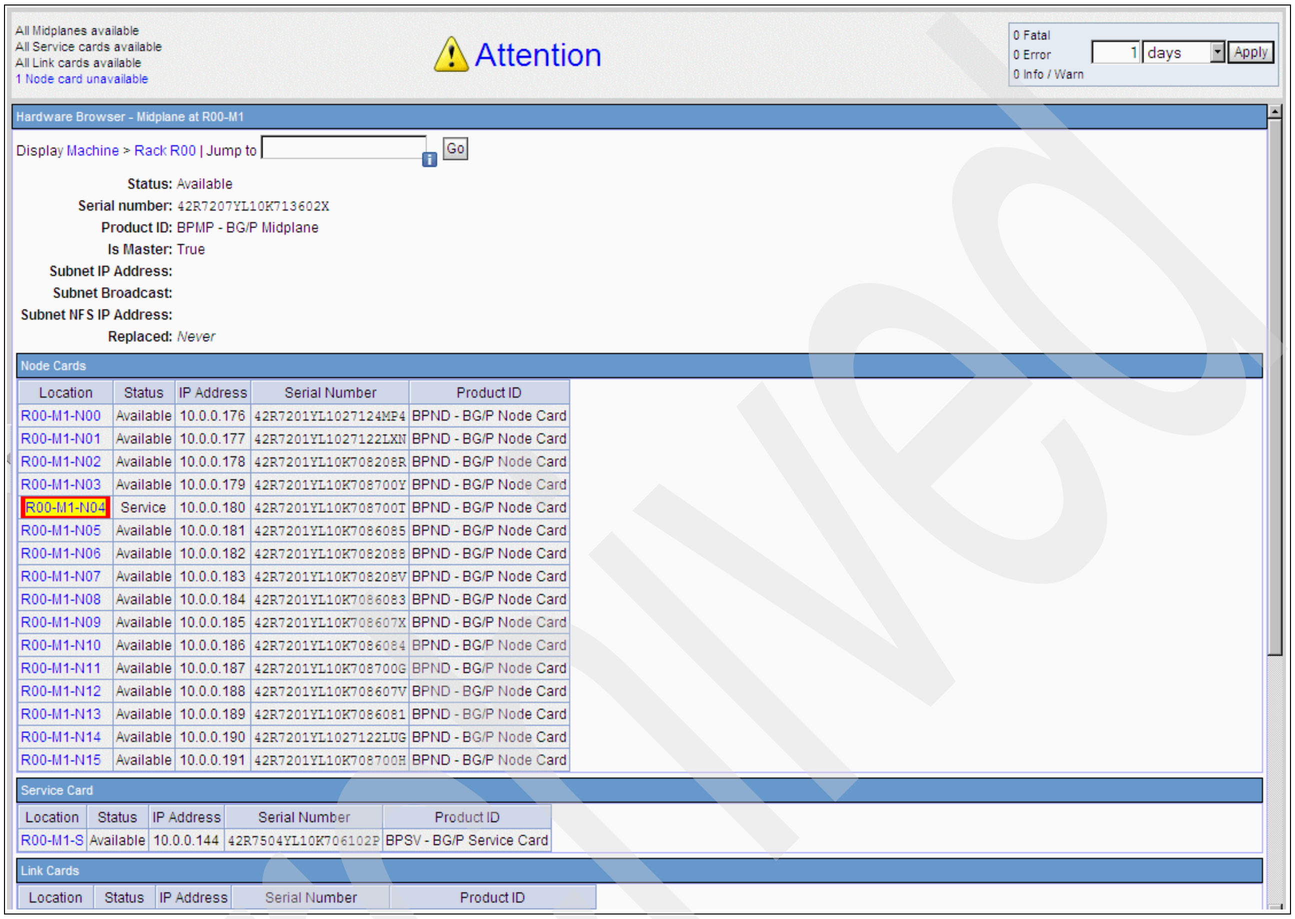
Clock Cable Connections

Clock source: R20

This rack does not propagate the clock signal.

Figure 2-47 Rack information

Click one of the midplane links to display all of the cards in that midplane. Again, we clicked the highlighted midplane to find the source of the unavailable hardware (see Figure 2-48). This page contains information about the node cards, Service card, Link cards and Fan Modules in the midplane. The unavailable hardware is node card R00-M1-N04, which has a pending Service Action.


All Midplanes available
All Service cards available
All Link cards available
1 Node card unavailable

Attention

0 Fatal
0 Error
0 Info / Warn

1 days Apply

Hardware Browser - Midplane at R00-M1

Display Machine > Rack R00 | Jump to Go

Status: Available
Serial number: 42R7207YL10K713602X
Product ID: BPMP - BG/P Midplane
Is Master: True
Subnet IP Address:
Subnet Broadcast:
Subnet NFS IP Address:
Replaced: *Never*

Node Cards

Location	Status	IP Address	Serial Number	Product ID
R00-M1-N00	Available	10.0.0.176	42R7201YL1027124MP4	BPND - BG/P Node Card
R00-M1-N01	Available	10.0.0.177	42R7201YL1027122LXN	BPND - BG/P Node Card
R00-M1-N02	Available	10.0.0.178	42R7201YL10K708208R	BPND - BG/P Node Card
R00-M1-N03	Available	10.0.0.179	42R7201YL10K708700Y	BPND - BG/P Node Card
R00-M1-N04	Service	10.0.0.180	42R7201YL10K708700T	BPND - BG/P Node Card
R00-M1-N05	Available	10.0.0.181	42R7201YL10K7086085	BPND - BG/P Node Card
R00-M1-N06	Available	10.0.0.182	42R7201YL10K7082088	BPND - BG/P Node Card
R00-M1-N07	Available	10.0.0.183	42R7201YL10K708208V	BPND - BG/P Node Card
R00-M1-N08	Available	10.0.0.184	42R7201YL10K7086083	BPND - BG/P Node Card
R00-M1-N09	Available	10.0.0.185	42R7201YL10K708607X	BPND - BG/P Node Card
R00-M1-N10	Available	10.0.0.186	42R7201YL10K7086084	BPND - BG/P Node Card
R00-M1-N11	Available	10.0.0.187	42R7201YL10K708700G	BPND - BG/P Node Card
R00-M1-N12	Available	10.0.0.188	42R7201YL10K708607V	BPND - BG/P Node Card
R00-M1-N13	Available	10.0.0.189	42R7201YL10K7086081	BPND - BG/P Node Card
R00-M1-N14	Available	10.0.0.190	42R7201YL1027122LUG	BPND - BG/P Node Card
R00-M1-N15	Available	10.0.0.191	42R7201YL10K708700H	BPND - BG/P Node Card

Service Card

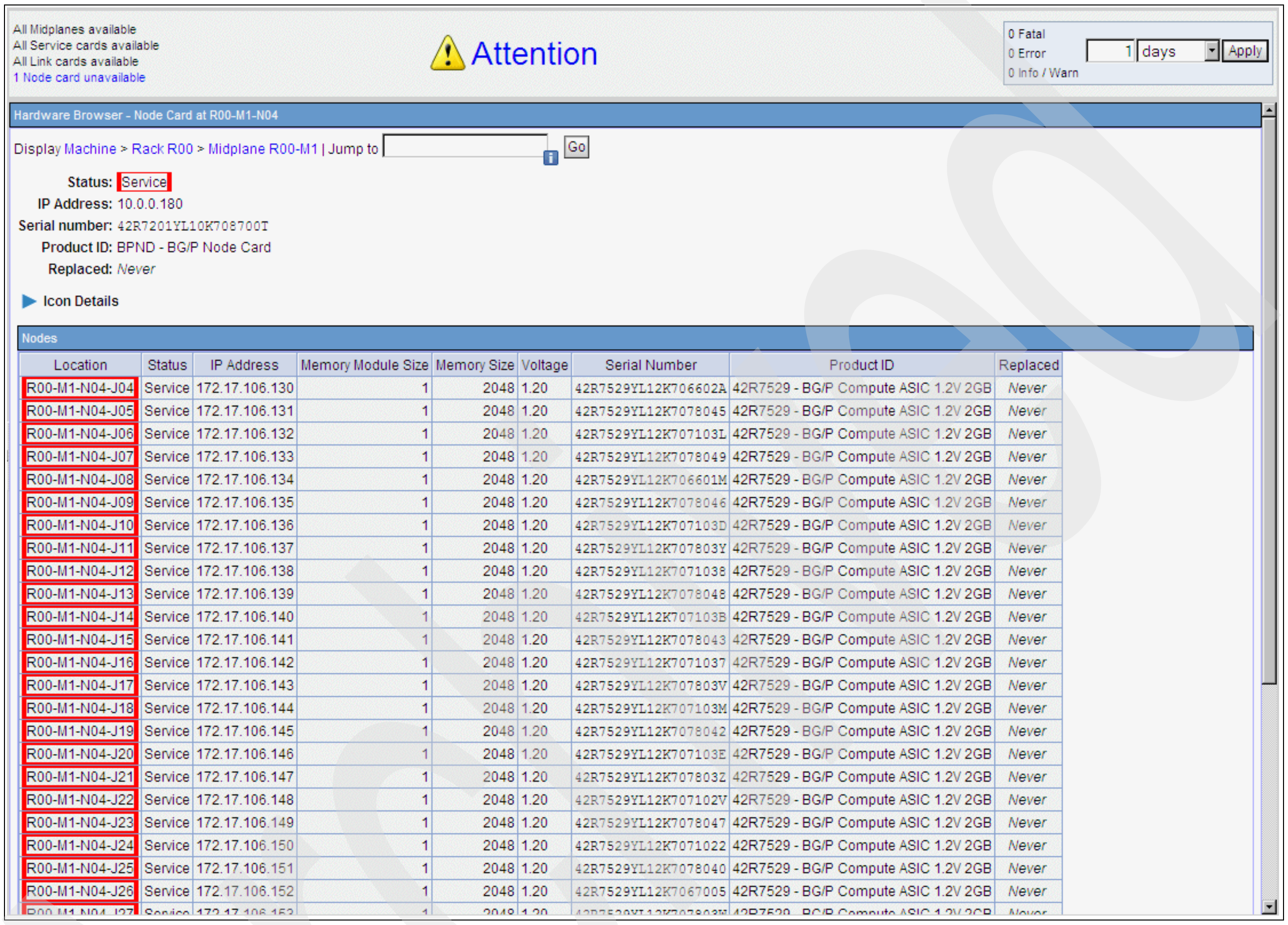
Location	Status	IP Address	Serial Number	Product ID
R00-M1-S	Available	10.0.0.144	42R7504YL10K706102P	BPSV - BG/P Service Card

Link Cards

Location	Status	IP Address	Serial Number	Product ID

Figure 2-48 Hardware browser midplane level

Clicking the available links drills down further on each of the cards. For example, if you click one of the node cards, you open a page similar to the one shown in Figure 2-49. In this case we continued following the path of the unavailable hardware. This page contains information about the status and address of each of the nodes, both compute and I/O nodes, on the node card. Also, the page contains information about the amount of memory on each of the nodes, voltages, identification number, and whether the card in this location has ever been replaced.


All Midplanes available
All Service cards available
All Link cards available
1 Node card unavailable
Attention
0 Fatal
0 Error
0 Info / Warn
1 days
Apply
Hardware Browser - Node Card at R00-M1-N04
Display Machine > Rack R00 > Midplane R00-M1 | Jump to
Go

Status: Service
IP Address: 10.0.0.180
Serial number: 42R7201YL10K708700T
Product ID: BPND - BG/P Node Card
Replaced: *Never*

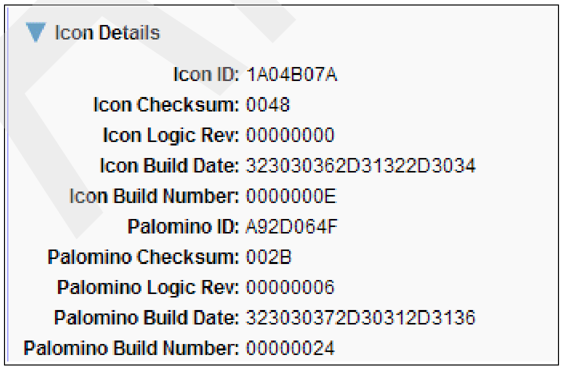
Icon Details

Nodes

Location	Status	IP Address	Memory Module Size	Memory Size	Voltage	Serial Number	Product ID	Replaced
R00-M1-N04-J04	Service	172.17.106.130	1	2048	1.20	42R7529YL12K706602A	42R7529 - BG/P Compute ASIC 1.2V 2GB	*Never*
R00-M1-N04-J05	Service	172.17.106.131	1	2048	1.20	42R7529YL12K7078045	42R7529 - BG/P Compute ASIC 1.2V 2GB	*Never*
R00-M1-N04-J06	Service	172.17.106.132	1	2048	1.20	42R7529YL12K707103L	42R7529 - BG/P Compute ASIC 1.2V 2GB	*Never*
R00-M1-N04-J07	Service	172.17.106.133	1	2048	1.20	42R7529YL12K7078049	42R7529 - BG/P Compute ASIC 1.2V 2GB	*Never*
R00-M1-N04-J08	Service	172.17.106.134	1	2048	1.20	42R7529YL12K706601M	42R7529 - BG/P Compute ASIC 1.2V 2GB	*Never*
R00-M1-N04-J09	Service	172.17.106.135	1	2048	1.20	42R7529YL12K7078046	42R7529 - BG/P Compute ASIC 1.2V 2GB	*Never*
R00-M1-N04-J10	Service	172.17.106.136	1	2048	1.20	42R7529YL12K707103D	42R7529 - BG/P Compute ASIC 1.2V 2GB	*Never*
R00-M1-N04-J11	Service	172.17.106.137	1	2048	1.20	42R7529YL12K707803Y	42R7529 - BG/P Compute ASIC 1.2V 2GB	*Never*
R00-M1-N04-J12	Service	172.17.106.138	1	2048	1.20	42R7529YL12K7071038	42R7529 - BG/P Compute ASIC 1.2V 2GB	*Never*
R00-M1-N04-J13	Service	172.17.106.139	1	2048	1.20	42R7529YL12K7078048	42R7529 - BG/P Compute ASIC 1.2V 2GB	*Never*
R00-M1-N04-J14	Service	172.17.106.140	1	2048	1.20	42R7529YL12K707103B	42R7529 - BG/P Compute ASIC 1.2V 2GB	*Never*
R00-M1-N04-J15	Service	172.17.106.141	1	2048	1.20	42R7529YL12K7078043	42R7529 - BG/P Compute ASIC 1.2V 2GB	*Never*
R00-M1-N04-J16	Service	172.17.106.142	1	2048	1.20	42R7529YL12K7071037	42R7529 - BG/P Compute ASIC 1.2V 2GB	*Never*
R00-M1-N04-J17	Service	172.17.106.143	1	2048	1.20	42R7529YL12K707803V	42R7529 - BG/P Compute ASIC 1.2V 2GB	*Never*
R00-M1-N04-J18	Service	172.17.106.144	1	2048	1.20	42R7529YL12K707103M	42R7529 - BG/P Compute ASIC 1.2V 2GB	*Never*
R00-M1-N04-J19	Service	172.17.106.145	1	2048	1.20	42R7529YL12K7078042	42R7529 - BG/P Compute ASIC 1.2V 2GB	*Never*
R00-M1-N04-J20	Service	172.17.106.146	1	2048	1.20	42R7529YL12K707103E	42R7529 - BG/P Compute ASIC 1.2V 2GB	*Never*
R00-M1-N04-J21	Service	172.17.106.147	1	2048	1.20	42R7529YL12K707803Z	42R7529 - BG/P Compute ASIC 1.2V 2GB	*Never*
R00-M1-N04-J22	Service	172.17.106.148	1	2048	1.20	42R7529YL12K707102V	42R7529 - BG/P Compute ASIC 1.2V 2GB	*Never*
R00-M1-N04-J23	Service	172.17.106.149	1	2048	1.20	42R7529YL12K7078047	42R7529 - BG/P Compute ASIC 1.2V 2GB	*Never*
R00-M1-N04-J24	Service	172.17.106.150	1	2048	1.20	42R7529YL12K7071022	42R7529 - BG/P Compute ASIC 1.2V 2GB	*Never*
R00-M1-N04-J25	Service	172.17.106.151	1	2048	1.20	42R7529YL12K7078040	42R7529 - BG/P Compute ASIC 1.2V 2GB	*Never*
R00-M1-N04-J26	Service	172.17.106.152	1	2048	1.20	42R7529YL12K7067005	42R7529 - BG/P Compute ASIC 1.2V 2GB	*Never*

Figure 2-49 Hardware Browser node card

Click the icon details twisty to open the section shown in Figure 2-50. This information pertains to the Icon and Palomino chips that are located on the card.



Icon Details

Icon ID:	1A04B07A
Icon Checksum:	0048
Icon Logic Rev:	00000000
Icon Build Date:	323030362D31322D3034
Icon Build Number:	0000000E
Palomino ID:	A92D064F
Palomino Checksum:	002B
Palomino Logic Rev:	00000006
Palomino Build Date:	323030372D30312D3136
Palomino Build Number:	00000024

Figure 2-50 Icon Details

2.3.12 Link Summary

Clicking **Link Summary** in the navigation pane shows a diagram of the links that are created with data cables in the X, Y, and Z dimensions. Figure 2-51 shows the links in a four rack configuration. These diagrams are especially useful when creating blocks, because you can see the progression of the cables through the rows and columns.

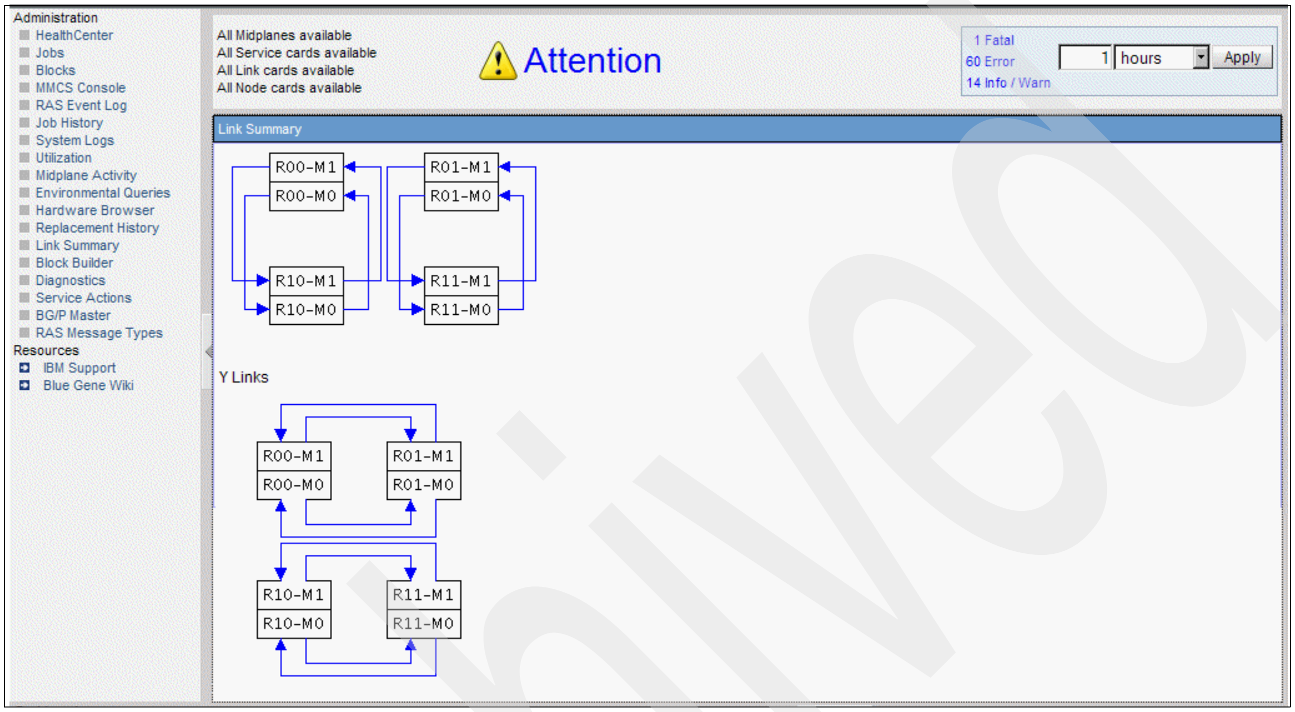
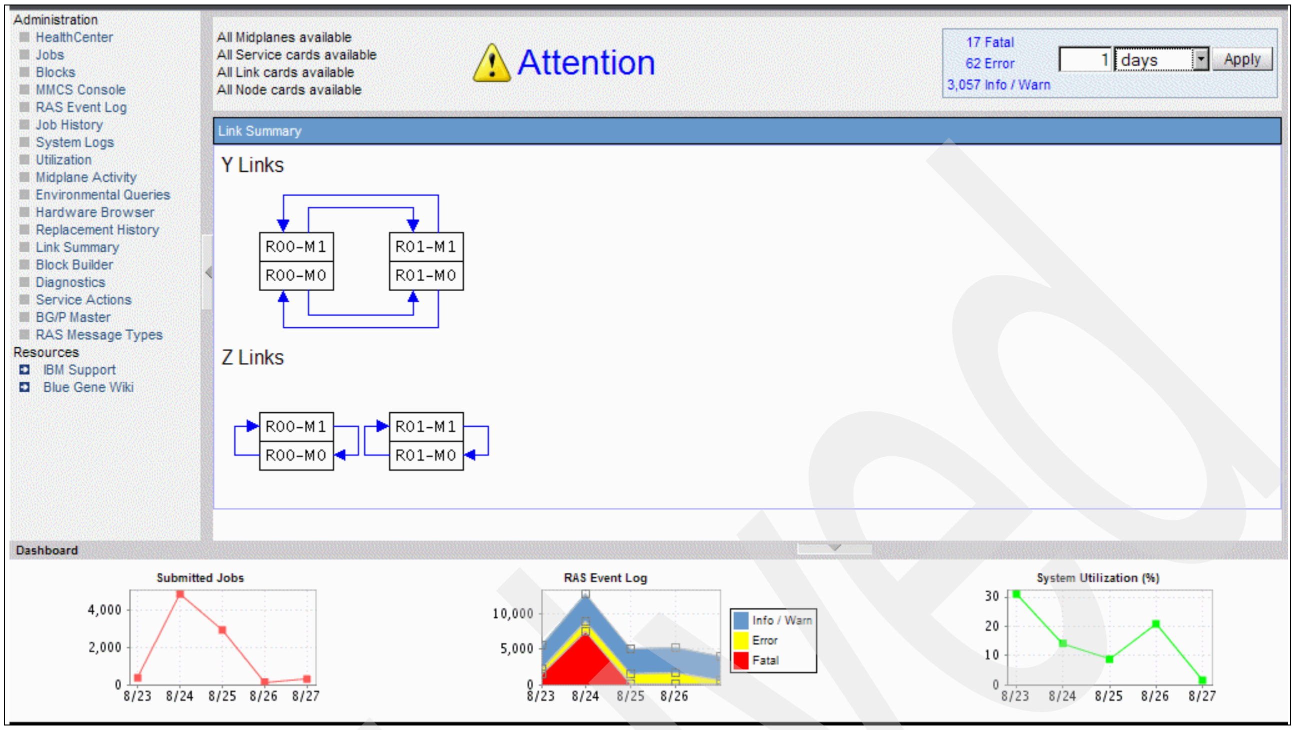


Figure 2-51 X and Y Links

Figure 2-52 shows how the Y and Z dimensions are cabled in a 2-rack system.

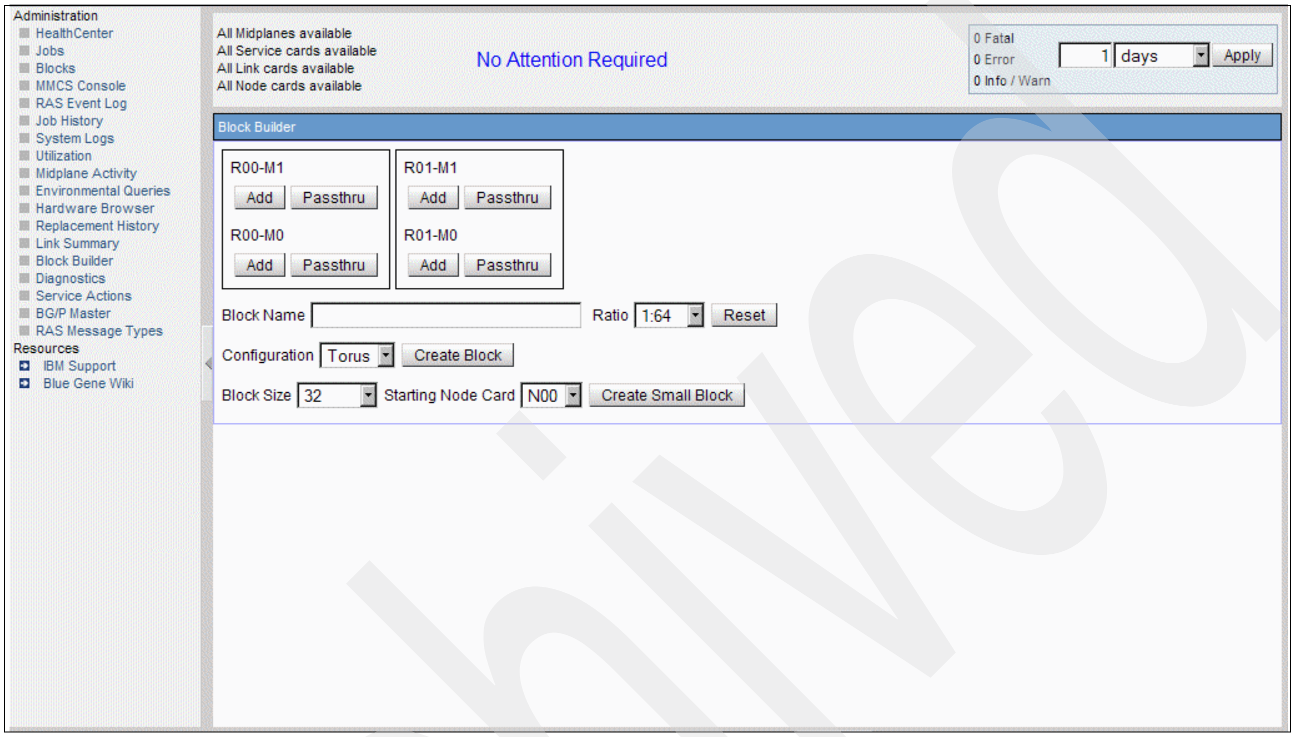


All Midplanes available
All Service cards available
All Link cards available
All Node cards available

Figure 2-52 Y and Z Links

2.3.13 Block Builder

Figure 2-53 shows the Block Builder page. This page provides a graphical interface to create any block that is needed on a Blue Gene/P system. For instructions about creating blocks using this tool, see 4.2, "Displaying and creating blocks using Blue Gene Navigator" on page 103.



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

No Attention Required

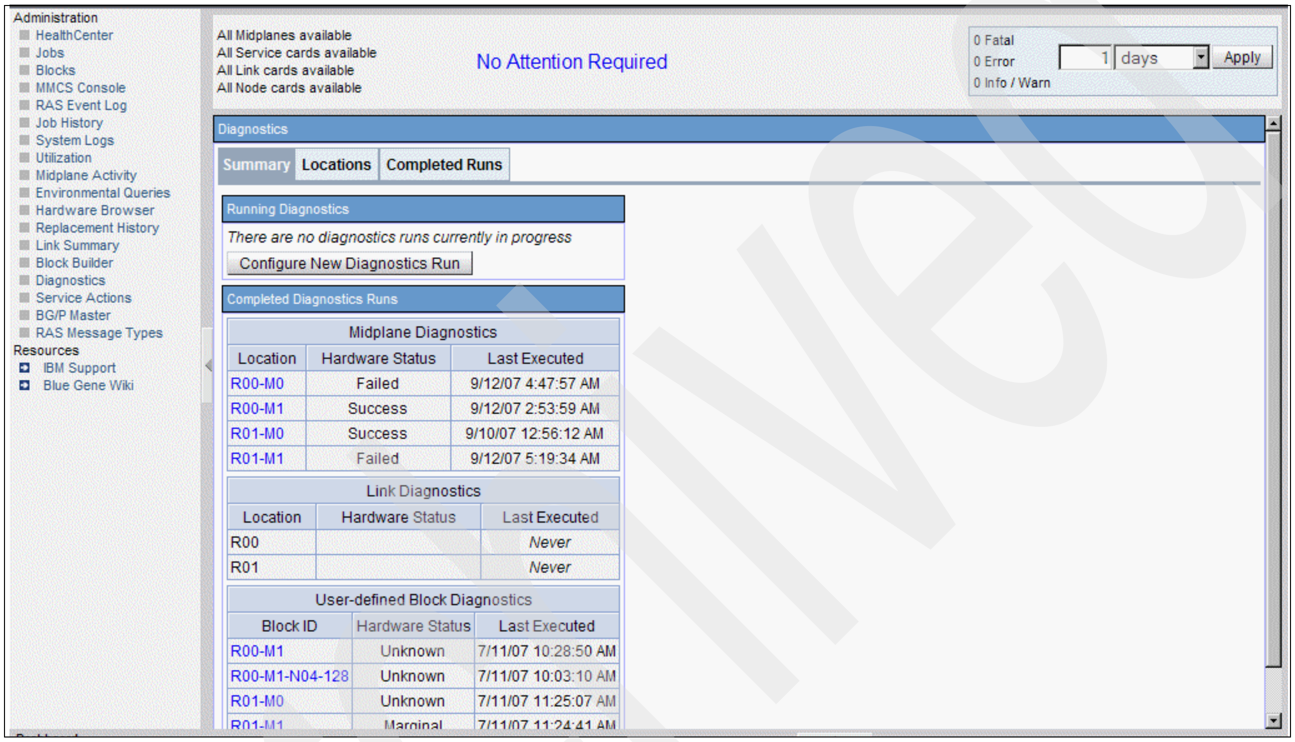
0 Fatal
0 Error
0 Info / Warn
1 days

Block Name
Ratio 1:64
Configuration Torus
Block Size 32
Starting Node Card N00

Figure 2-53 Block Builder

2.3.14 Diagnostics

The Diagnostics page, shown in Figure 2-54, provides a method to create new Diagnostic runs and also view runs that are currently running or have completed. The Summary tab shows any Diagnostic runs that are active and also the status of the most recent completed Midplane, Link, and User-defined Block run. You can also initiate a new run from this tab. The Locations and Completed Runs tabs allow you to view previous runs. The Filter Options on the Locations tab allows you to narrow the results based on location, hardware status, time of the run or whether hardware was replaced. On the Completed Runs tab, you can filter on the status of the runs, time, midplane, racks, or specific blocks.



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

No Attention Required

Running Diagnostics

There are no diagnostics runs currently in progress

Completed Diagnostics Runs

Midplane Diagnostics

Location	Hardware Status	Last Executed
R00-M0	Failed	9/12/07 4:47:57 AM
R00-M1	Success	9/12/07 2:53:59 AM
R01-M0	Success	9/10/07 12:56:12 AM
R01-M1	Failed	9/12/07 5:19:34 AM

Link Diagnostics

Location	Hardware Status	Last Executed
R00		*Never*
R01		*Never*

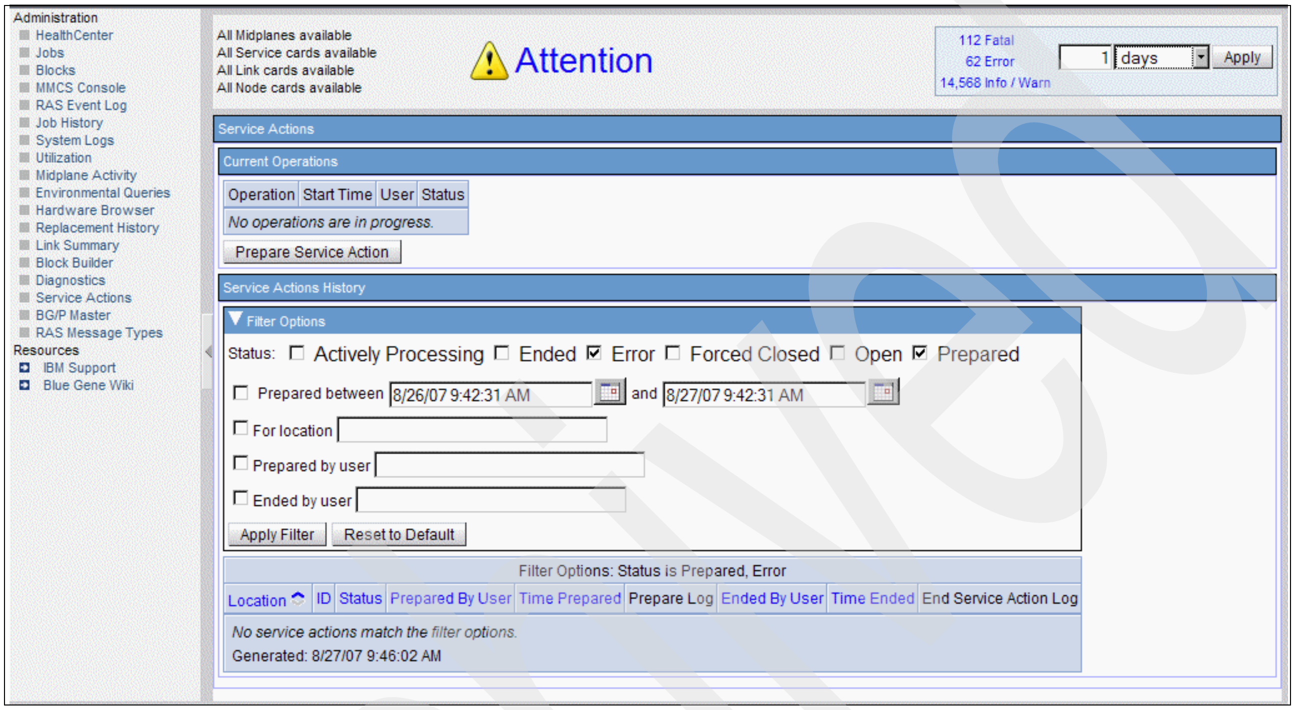
User-defined Block Diagnostics

Block ID	Hardware Status	Last Executed
R00-M1	Unknown	7/11/07 10:28:50 AM
R00-M1-N04-128	Unknown	7/11/07 10:03:10 AM
R01-M0	Unknown	7/11/07 11:25:07 AM
R01-M1	Marginal	7/11/07 11:24:41 AM

Figure 2-54 Diagnostics home page

2.3.15 Service Actions

Navigator allows administrators to initiate Service Actions from the browser. You can also query the database through the Navigator to view current or historical Service Actions. The Filter Options also allow you to limit query results by time stamp, user, or location. Figure 2-55 shows the Service Action page with the Filter Options expanded. More information about Service Actions is available in Chapter 11, “Service Actions” on page 199.



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

Service Actions

Current Operations

Operation	Start Time	User	Status
No operations are in progress.			

Prepare Service Action

Service Actions History

Filter Options

Status: ☐ Actively Processing ☐ Ended ☑ Error ☐ Forced Closed ☐ Open ☑ Prepared

☐ Prepared between 8/26/07 9:42:31 AM and 8/27/07 9:42:31 AM

☐ For location

☐ Prepared by user

☐ Ended by user

Apply Filter | Reset to Default

Filter Options: Status is Prepared, Error

Location	ID	Status	Prepared By User	Time Prepared	Prepare Log	Ended By User	Time Ended	End Service Action Log

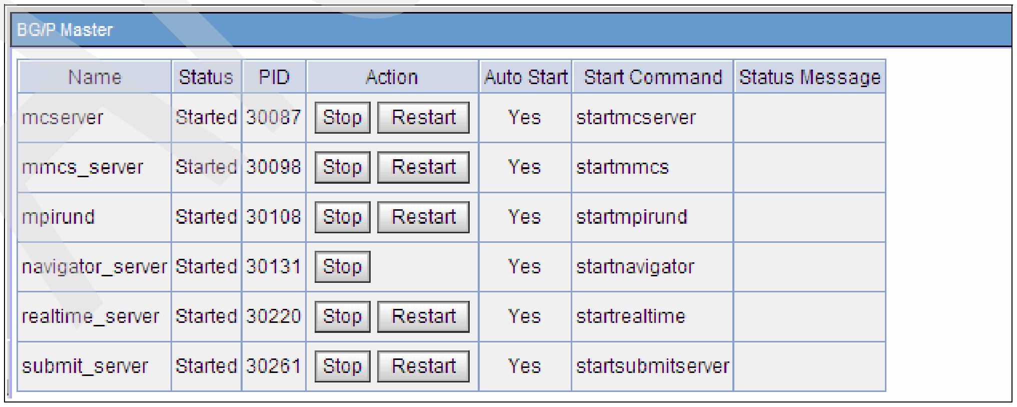
No service actions match the filter options.

Generated: 8/27/07 9:46:02 AM

Figure 2-55 Blue Gene/P Service Actions

2.3.16 The BG/P Master page

Figure 2-56 shows the Navigator’s interface to the BGPMaster processes, the BG/P Master page. You can start each of the servers that are monitored by BGPMaster, except navigator_server, from this page.



BG/P Master

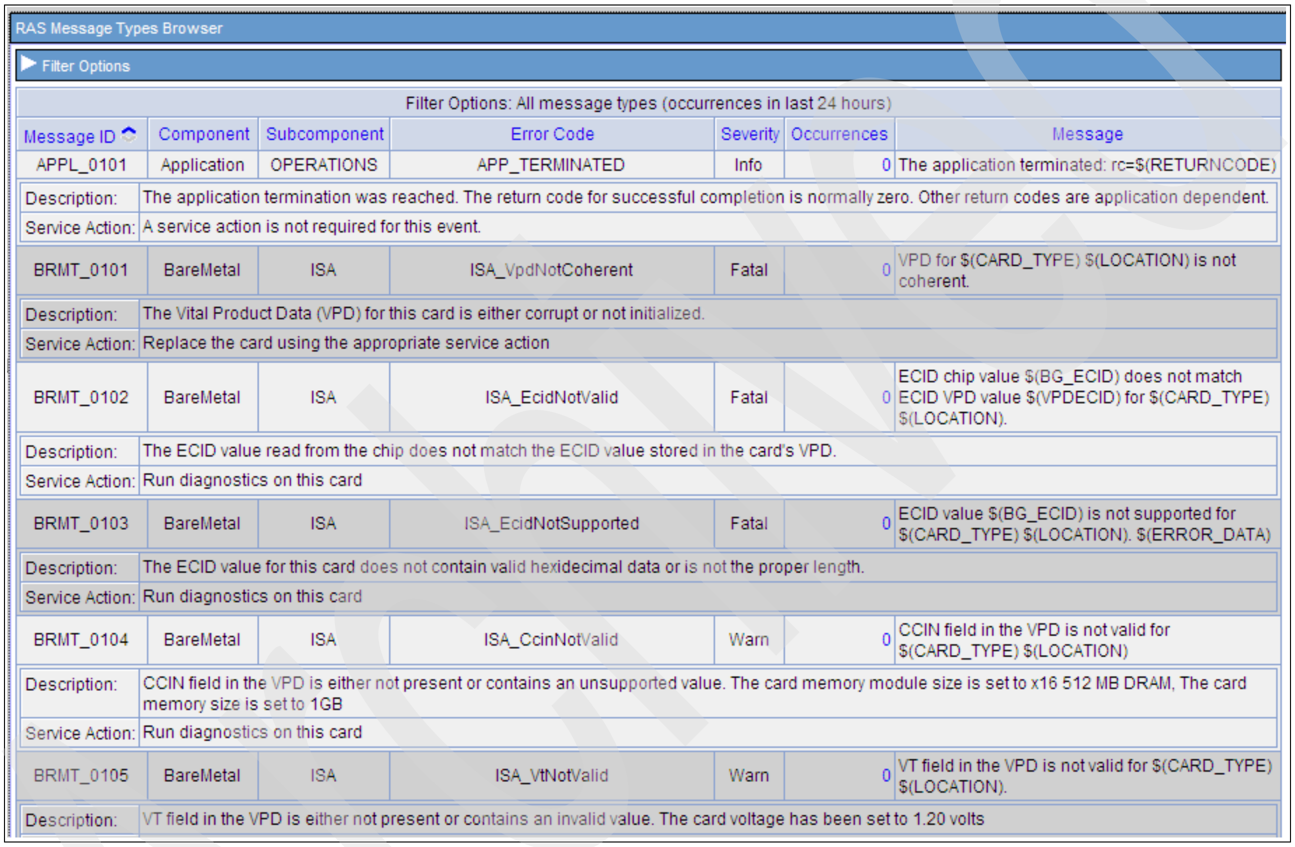
Name	Status	PID	Action	Auto Start	Start Command	Status Message
mcserver	Started	30087	Stop Restart	Yes	startmcserver	
mmcs_server	Started	30098	Stop Restart	Yes	startmmcs	
mpirund	Started	30108	Stop Restart	Yes	startmpirund	
navigator_server	Started	30131	Stop	Yes	startnavigator	
realtime_server	Started	30220	Stop Restart	Yes	startrealtime	
submit_server	Started	30261	Stop Restart	Yes	startsubmitserver	

Figure 2-56 BG/P Master

You can stop all of the servers. However, keep in mind that if you stop the navigator_server, you no longer have a connection to restart the processes. In this case, you have to restart, at the very least, the navigator_server from the command line.

2.3.17 RAS Message Types

Figure 2-57 shows the RAS Message Type page. The default shows all messages, and the default time is set to the last 24-hour period. You can sort the results by clicking any of the column heads. The numbers in the *Occurrences* column are links. Clicking the links opens the RAS Events page that shows the actual events for the message type over the specified interval.

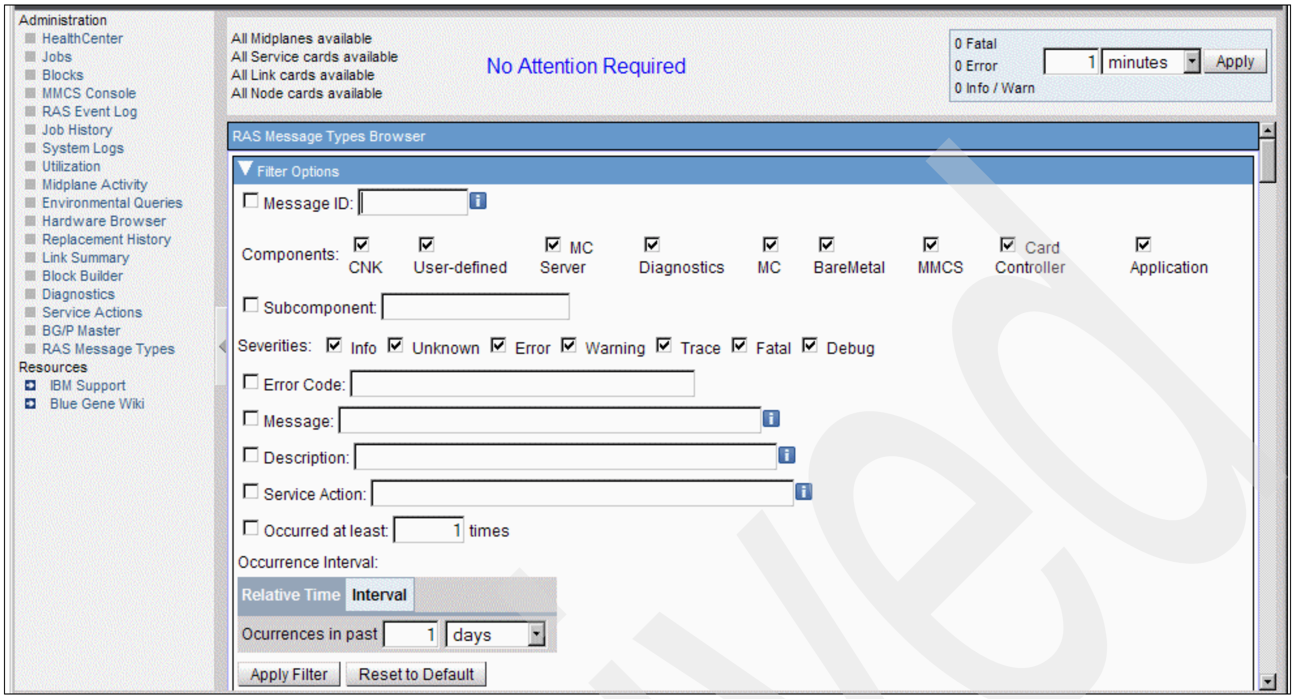


Filter Options: All message types (occurrences in last 24 hours)

Message ID	Component	Subcomponent	Error Code	Severity	Occurrences	Message
APPL_0101	Application	OPERATIONS	APP_TERMINATED	Info	0	The application terminated: rc=$(RETURNCODE)
Description:	The application termination was reached. The return code for successful completion is normally zero. Other return codes are application dependent.					
Service Action:	A service action is not required for this event.					
BRMT_0101	BareMetal	ISA	ISA_VpdNotCoherent	Fatal	0	VPD for $(CARD_TYPE) $(LOCATION) is not coherent.
Description:	The Vital Product Data (VPD) for this card is either corrupt or not initialized.					
Service Action:	Replace the card using the appropriate service action					
BRMT_0102	BareMetal	ISA	ISA_EcidNotValid	Fatal	0	ECID chip value $(BG_ECID) does not match ECID VPD value $(VPDECID) for $(CARD_TYPE) $(LOCATION).
Description:	The ECID value read from the chip does not match the ECID value stored in the card's VPD.					
Service Action:	Run diagnostics on this card					
BRMT_0103	BareMetal	ISA	ISA_EcidNotSupported	Fatal	0	ECID value $(BG_ECID) is not supported for $(CARD_TYPE) $(LOCATION). $(ERROR_DATA)
Description:	The ECID value for this card does not contain valid hexidecimal data or is not the proper length.					
Service Action:	Run diagnostics on this card					
BRMT_0104	BareMetal	ISA	ISA_CcinNotValid	Warn	0	CCIN field in the VPD is not valid for $(CARD_TYPE) $(LOCATION)
Description:	CCIN field in the VPD is either not present or contains an unsupported value. The card memory module size is set to x16 512 MB DRAM, The card memory size is set to 1GB					
Service Action:	Run diagnostics on this card					
BRMT_0105	BareMetal	ISA	ISA_VtNotValid	Warn	0	VT field in the VPD is not valid for $(CARD_TYPE) $(LOCATION).
Description:	VT field in the VPD is either not present or contains an invalid value. The card voltage has been set to 1.20 volts					

Figure 2-57 RAS Message Types

Figure 2-58 displays the possible Filter Options available in the RAS Message Types list.


Administration
- HealthCenter
- Jobs
- Blocks
- MMCS Console
- RAS Event Log
- Job History
- System Logs
- Utilization
- Midplane Activity
- Environmental Queries
- Hardware Browser
- Replacement History
- Link Summary
- Block Builder
- Diagnostics
- Service Actions
- BG/P Master
- RAS Message Types

Resources
- IBM Support
- Blue Gene Wiki

All Midplanes available
All Service cards available
All Link cards available
All Node cards available

No Attention Required

0 Fatal
0 Error
0 Info / Warn

1 minutes Apply

RAS Message Types Browser

Filter Options

☐ Message ID:

Components: ☑ CNK ☑ User-defined ☑ MC Server ☑ Diagnostics ☑ MC ☑ BareMetal ☑ MMCS ☑ Card Controller ☑ Application

☐ Subcomponent:

Severities: ☑ Info ☑ Unknown ☑ Error ☑ Warning ☑ Trace ☑ Fatal ☑ Debug

☐ Error Code:

☐ Message:

☐ Description:

☐ Service Action:

☐ Occurred at least: 1 times

Occurrence Interval:

Relative Time | Interval

Ocurrences in past 1 days

Apply Filter | Reset to Default

Figure 2-58 RAS Message Types Filter Options

The first option allows you to filter by *Message ID*. The field accepts up to 10 characters, and you can also use wildcards (* or ?). You can search the messages by *Components* or message *Severities* by selecting the source or level of message that you want to see. By default, all of the options are selected, and you can eliminate any of the options by deselecting them.

If you look back at Figure 2-57 on page 71, you see the *Error Code* column in the center of the page. The error code strings are also searchable by enabling the *Error Code* option and then supplying the string in the box that is provided. Similarly, you can use each of the next three options (*Message*, *Description* and *Service Action*) as a search argument by checking the appropriate box and providing a string. Each of the fields allow you to enter up to 1024 characters or wildcards.

The next option, *Occurred at least: x times*, provides a method of searching based on the number of occurrences a message has appeared. Selecting the box and providing a threshold value returns the number of times a message has been entered in the RAS database in the time period specified. Using this filter option gives you the opportunity to evaluate whether errors are random or if there might be a trend developing.

The last option is to set the time period to be searched. You can search back in time starting from the current time, which is the Relative Time tab. The other method is to click the Interval tab, which gives you the opportunity to supply a start and stop time to narrow your search. Your search can be broad by selecting only a component or a message severity, or it can be very defined by selecting more than one option and providing specific messages and time periods.

2.4 Customizing Blue Gene Navigator

As we mentioned in the previous sections, there are several areas of the Blue Gene Navigator interface that are customizable. In this section, we describe how to add to or change the interface to suit your environment.

When you install the system, the default location of the configuration file is `/bgsys/drivers/ppcfloor/navigator/navigator-config.xml`. Depending on your policy, you might want to create a local configuration file (`navigator-config.xml`), which is usually located in `/bgsys/local/etc`, with your additional configurations. Modifications made in this file will not be affected by driver upgrades.

2.4.1 Adding resource links

You can replace existing or define additional resource links to display in the navigation pane of the Navigator. Example 2-21 demonstrates how to add a new link to the navigation pane. You must supply the following values in the resource-link container in your local configuration file:

`name`	Must be unique for each definition. When overriding values for an existing definition the same name must be given.
`enabled`	A Boolean value that indicates whether the link displays. If true, then the link displays in the browser. The default is true.
`label`	Displays as the text of the resource link in the navigation pane.
`url`	The URL to which the browser is directed when the link is selected.
`new-page`	If set to `true`, a new page opens when the user clicks the link. The default is true.
`navigation-image`	Overrides the default bullet image. Should be a URL that the browser can use to fetch an 8x8 pixels image. Displays next to the link in the Navigation area.
`style`	Overrides the default style. Should be Cascading Style Sheets (CSS).
`priority`	Resource links displays in order of increasing priority, from first to last. This must be an integer.

Example 2-21 Resource-link container

```
<resource-link>
   <name>New Link</name>
   <enabled>true</enabled> <!--not required unless setting to ‘false’-->
   <label>My New Link</label>
   <url>http://www.NewLink.com</url>
   <new-page>true</new-page> <!--not required unless setting to false-->
   <navigation-image>http://example.com/image1.png</navigation-image>
   <style>color: green;</style>
   <priority>2500</priority>
</resource-link>
```

There is one default resource link that is defined in the Navigator configuration file that provides a link to the IBM Blue Gene support home page. Example 2-22 shows the configuration file contents for this link.

Example 2-22 Default resource link

```
<resource-link>
    <name>IBMSupport</name>
    <label>IBM Support</label>
    <url>http://www.ibm.com/servers/eserver/support/bluegene/index.html</url>
    <priority>1000</priority>
  </resource-link>
```

As an example of overriding a value of the default resource link, this link can be disabled by setting the `enabled` tag to `false` in the local configuration file (Example 2-23).

Example 2-23 Disabling a resource link

```
<resource-link>
    <name>IBMSupport</name>
    <enabled>false</enabled>
</resource-link>
```

2.4.2 Chart plug-ins

You can use two types of plug-ins to customize the Navigator interface for your environment:

- Attention Plug-ins cause the Attention indicator in the Attention area to display and to have messages display in the Health Center.
- Chart Plug-ins add charts to the Dashboard

The following chart plug-ins ship by default (Example 2-24):

- Submitted jobs
- Software RAS events
- System utilization

Example 2-24 Default plug-ins

```
<chart-plugin>
    <name>jobs</name>
    <class>com.ibm.bluegene.navigator.charts.JobsChartGenerator</class>
    <priority>1000</priority>
  </chart-plugin>
  <chart-plugin>
    <name>ras</name>
    <class>com.ibm.bluegene.navigator.charts.RasChartGenerator</class>
    <priority>2000</priority>
  </chart-plugin>
  <chart-plugin>
    <name>utilization</name>
    <class>com.ibm.bluegene.navigator.charts.UtilizationChartGenerator</class>
    <priority>3000</priority>
  </chart-plugin>
```

Creating a plug-in

Whether creating an attention plug-in or a chart plug-in, you need to override a class that is provided in `navigator.jar`, which is shipped in the following directory:

```
/bgsys/drivers/ppcfloor/navigator/catalina_base/webapps/BlueGeneNavigator/WEB-INF/lib
```

Build your classes with `navigator.jar` in the classpath. Then create a `.jar` file containing the classes and a definition file.

Attention plug-in

Attention plug-ins extend `com.ibm.bluegene.navigator.attention.AbstractAttentionPlugin`. When Navigator starts, it creates an instance of your class using the default constructor and then calls `initialize()` with a `com.ibm.bluegene.navigator.config.AttentionPluginConfig` object. From the `AttentionPluginConfig` passed to `initialize()`, you can get the properties that the user has configured for your plug-in.

Whenever a page is requested, the Navigator can call the `isAttentionRequired` method (it will not call it if the attention state is already known). This must return true to indicate that attention is required. Otherwise, false is returned.

When the Health Center page is requested, the Navigator calls the getAttentionMessages method that should return a list of `AbstractAttentionPlugin.MessageInfo` objects. The `MessageInfos` returned displays in the Health Center. The returned messages are labeled with the string returned by `getLabel`.

The attention plug-in's definition file must have an `attention-plugin` element with a unique name and reference your implementation class. Example 2-25 gives an example `attention-plugin` configuration file.

Example 2-25 An attention-plugin file

```
<?xml version="1.0" standalone="yes">

<navigator>
  <attention-plugin>
    <name>Sample1</name>
    <class>com.example.plugin</class>
  </attention-plugin>
</navigator>
```

Chart plug-in

Chart plug-ins extend `com.ibm.bluegene.navigator.charts.AbstractChartGenerator`.

When Navigator starts, it creates an instance of your class using the default constructor and then calls `initialize()` with a `com.ibm.bluegene.navigator.config.ChartPluginConfig` object. From the `AttentionPluginConfig` passed to `initialize()`, you can get the properties that the user has configured for your plug-in. The methods that you want to override are `getLabel` and `getChartInfo`. `getLabel` supplies Navigator with the text to display above the chart in the Dashboard. `getChartInfo` can be called every time a request is made to Navigator (it will not be called if the Dashboard is hidden) and supplies Navigator with the information that it sends to the client so that the client will request the image. You might also want to override `getLegendHtml` to provide a legend to be displayed next to the chart.

For most types of charts, it will probably not be fast enough to generate the image on every request. For this reason, a class is supplied that generates the image in a background thread every once a while and just sends back the cached result on each request. This class is `com.ibm.bluegene.navigator.config.AbstractTimeoutThreadChartGenerator` which extends `AbstractChartGenerator`.

To use this class, you override several abstract methods:

`initializeInstance()`	Called when Navigator starts up. You can get any property values from the supplied `ChartPluginConfig`.
`generateChartInfo()`	Called from a thread when the timeout occurs to generate the new chart info.
`applyState()`	Called with the result of `generateChartInfo()`.
`getReqChart()`	Called whenever a page is requested to return the cached `ChartInfo`. You will probably get the cached `ChartInfo` from what was set by `applyState()`.
`getThreadName()`	Used when debugging to make it easy to find the thread that is calling `generateChartInfo` and `applyState`.

You still want to override `getLabel`, and you might want to override `getLegendHtml`.

Installing Navigator plug-ins

There are two parts to the plug-in:

- A definition file
- A `.jar` file

The definition file is named `navigator-config.`*`<id>`*`.xml`, where *<id>* is a unique name. Put this file in *<configpath>* (`/bgsys/local/etc/` by default). The `.jar` file is put into *<navigatorlibs>* (`/bgsys/local/lib/navigator` by default). Make sure that Navigator can read these files.

Configuring Navigator plug-ins

Take a look at the plug-in's configuration file in *<configpath>* (`/bgsys/local/etc/` by default). If you want to override any options, you need to know the type and name of the plug-in, which you can determine from its configuration file:

- The type is indicated by whether the configuration file contains an `attention-plugin` or a `chart-plugin` element.
- The name is in the name element under that element. To override an option, add the same `attention-plugin` or `chart-plugin` element to your local Navigator configuration file *<configpath>*`/navigator-config.xml` (`/bgsys/local/etc/navigator-config.xml` by default) with the new value for the option.

The options for both types of plug-ins are:

`enabled`	To disable, set to `false`. If disabled the plug-in is not loaded. Defaults to `true`.
`class`	The fully-qualified name of the class that implements the plug-in.

Plug-ins can also define properties to define other options. Properties are defined in property elements (Example 2-26).

Example 2-26   Property elements

```
<chart-plugin>
  <property>
    <name>prop-name</name>
    <value>prop-value</value>
  </property>
</chart-plugin>
```

Additional options for chart plug-ins are:

`priority`	Defines the order that the chart displays in the dashboard frame. The value `0`, the default, is at the left side of the frame.
`timeout`	The time between chart generations, or use the default.

As an example of overriding an option for a plug-in, consider setting the priority to `15` for a chart plug-in whose name is `Sample1`. The local Navigator configuration file is similar to that shown in Example 2-27.

Example 2-27   Local configuration file

```
<navigator>
  <chart-plugin>
    <name>Sample1</name>
    <priority>15</priority>
  </chart-plugin>
</navigator>
```

More resources

The JavaDocs for classes that are used by plug-ins ship in `navigator/doc/plugin/javadoc`.

A sample `attention-plugin` file ships in `navigator/doc/plugin/sample/attention_plugin`. The `DiagsIntervalAttentionPlugin` causes the Attention indicator to display if diagnostics have not run successfully on a midplane in some interval (the default is 1 month). The interval can be overridden by setting the `interval` property for the plug-in. The value of the `interval` property must be an SQL Labeled Duration (for example, `1 MONTH`).

A sample chart plug-in is shipped in `navigator/doc/plugin/sample/chart_plugin`. The `NodeTempChartPlugin` charts the maximum temperature of any node for each day for the past 5 days. It uses JFreeChart to generate the image.

2.4.3 Problem determination

This section covers a few of the more common issues that arise with Navigator.

Navigator logging level

Navigator uses three loggers:

- The `com.ibm.bluegene.navigator` is used for general logging messages.
- The `com.ibm.bluegene.navigator.db` is used for database queries. Set the level for `com.ibm.bluegene.navigator.db` to `FINE`, and the database statements that are used are logged.
- The `com.ibm.bluegene.navigator.mmcs` logger is for MMCS Console log messages. Set the logging level for `com.ibm.bluegene.navigator.mmcs` to `FINE`, and the commands that are sent to mmcs_db_server are logged.

You can set the logging level in two ways:

- *Static log level* setting requires that you restart the Navigator, and the logging level continues to be used across Navigator restarts.
- *Dynamic log level* settings are made while Navigator is running and take place instantly, but the setting is discarded when Navigator is restarted.

To use the static method, edit the file /bgsys/drivers/ppcfloor/navigator/logging.properties and set the logging levels for the different loggers. For example, to set the logging level for the com.ibm.bluegene.navigator logger, remove the number sign (`#`) from in front of the line for `com.ibm.bluegene.navigator.level`, and set the logging level after the equal sign (`=`).

To use the dynamic method, change the last part of the URL to `about.jsp`, and press Enter to load the page. Expand the Logging Configuration section, and you can set the logging level for each logger using the drop-down menu. Then, select **Apply**.

Authentication issues

If users cannot authenticate, then usually one of the following issues is occurring:

- Users are not in the correct group.
- The user under which Tomcat is running does not have the correct authority.

To check whether users are in the correct group, look in the local navigator configuration file (`/bgsys/local/etc/navigator-config.xml` by default), and make sure that the correct groups are listed.

To check whether the user under which Tomcat is running has the correct authority, look near the top of the Navigator log file for the groups for which the user that started Navigator is a member. Make sure that the user or group has the authority that is required to do PAM authentication. The most common PAM setup uses the unix2 PAM module and thus requires that the user can read `/etc/shadow`. (Note that if you change the groups for a user, the change is not picked up by existing sessions. Thus, you must sign off before restarting Navigator and then check the Navigator log file again.)

Make sure that the user is in the correct groups. Otherwise, the user will be rejected. Check the group membership using `groups <username>`. Then, look in the local Navigator configuration file (`/bgsys/local/etc/navigator-config.xml` by default), and make sure that one of the groups of which the user is a member is listed in the group settings. Check `/var/log/messages` to make sure that PAM is not printing out something.

JAAS tracing

If the group membership is correct and the users still cannot authenticate, you need to check that users are not being rejected during the JAAS step. To get the trace messages for the JAAS module, edit the following file:

```
/bgsys/drivers/ppcfloor/navigator/catalina_base/conf/login.config
```

Add `debug=true` before the semicolon (;), as shown in Example 2-28. The debug text is printed to the regular Navigator log.

Example 2-28 Setting debug mode

```
BlueGeneNavigator {
  com.ibm.bluegene.auth.module.LinuxPAMLoginModule required
config="/bgsys/local/etc/navigator-config.xml,/bgp/dist/etc/navigator-config.xml"
    library="/bgsys/drivers/ppcfloor/navigator/bluegenenavigator.so" debug=true;
};
```

3

BGPMaster

BGPMaster is the process that monitors the following Blue Gene/P control processes:

- mcServer
- Midplane Management Control System (MMCS)
- mpirund
- Blue Gene Navigator
- Real-time Server
- Submit Server
- Event Log Analysis (ELA) Server

In this chapter, we explain the function of the BGPMaster and how to start and stop the BGPMaster process and the jobs that run under it. We also examine the logs to which each of the jobs writes.

3.1 BGPMaster

BGPMaster is a daemon running on the service node that monitors and restarts any failed control system components. By design BGPMaster can control the following components:

- mcServer
- mmcs_db_server
- mpirund
- navigator_server
- submit_server
- realtime_server

To ensure proper authority to log files, a member of the bgpadmin group needs to start BGPMaster.

3.1.1 BGPMaster configuration file

When the BGPMaster startup script is initialized, it reads from the `bgpmaster.init` file. By default, the script looks in the `/bgsys/local/etc` directory for any local configurations. You need to make changes to the default characteristics in the `.init` file in that directory so that release upgrades will not affect the expected behavior in your environment.

Example 3-1 shows the contents of a `bgpmaster.init` file. Commenting out the start line for a server (that is, inserting a number sign, #, at the beginning of the line) causes the startup process to skip the server on that line.

Example 3-1 The bgpmaster.init file

```
# This bgpmaster.init file has lines of the format
#
#     # comment
#     define server-name start-command
#     start  server-name
#
# The servers are initially started in the order that they are defined
#
define    mcserver         startmcserver
define    mmcs_server      startmmcs
define    mpirund          startmpirund
define    navigator_server startnavigator
define    realtime_server  startrealtime
define    submit_server    startsubmitserver
define    ela_server       startelaserver

start     mcserver
start     mmcs_server
start     mpirund
start     navigator_server
start     realtime_server
start     submit_server
# start   ela_server
```

You can find a template of the init file in the `/bgsys/drivers/ppcfloor/bin` directory. The name of the file is `bgpmaster.init.tpl`.

3.1.2 BGPMaster command

The **bgpmaster** command is used to start, stop, or restart BGPMaster or any of the individual servers that it monitors. You can also use this command to check the status of the BGPMaster daemon and the individual servers that it is monitoring. You must run the command from the `/bgsys/driver/ppcfloor/bin` directory.

The **bgpmaster** command uses the following syntax:

```
bgpmaster <command> [options]
```

Table 3-1 lists the required <*command*> parameter options.

Table 3-1 Command parameters for the bgpmaster command

Command	Syntax	Description
`start`	`bgpmaster start [`*server-name*`]`	If `start` is the only parameter supplied, the BGPMaster daemon is started along with any of the servers specified in the `bgpmaster.init` file. If a server name is specified and that server is not already running, it is started.
`stop`	`bgpmaster stop [`*server-name*`]`	If `stop` is the only parameter supplied, the BGPMaster daemon is stopped along with any of the servers it has started. If a server name is specified and that server is running, it is stopped.
`restart`	`bgpmaster restart [`*server-name*`]`	The `restart` parameter stops and restarts the BGPMaster process and all servers specified in the `bgpmaster.init` file. If an individual server name is provided, only that specific server is stopped and restarted.
`reload`	`bgpmaster reload`	Tells the BGPMaster daemon to reload and process the `bgpmaster.init` file.
`status`	`bgpmaster status`	Returns the current status (stopped or started) of all servers specified in the `bgpmaster.init` file.

Table 3-2 lists the available options for the **bgpmaster** command.

Table 3-2 Options for the bgpmaster command

Option	Syntax	Description
--autorestart	<y or n>	Determines if **bgpmaster** restarts failed servers automatically. If **n** is specified, a failed server must be restarted manually.
--host	<hostname or IP address>	Specifies the host name or IP address of the **bgpmaster** daemon. Default is 127.0.0.1.
--port	<IP port>	Specifies the port on which the **bgpmaster** daemon listens.
--help	none	Provides help text.

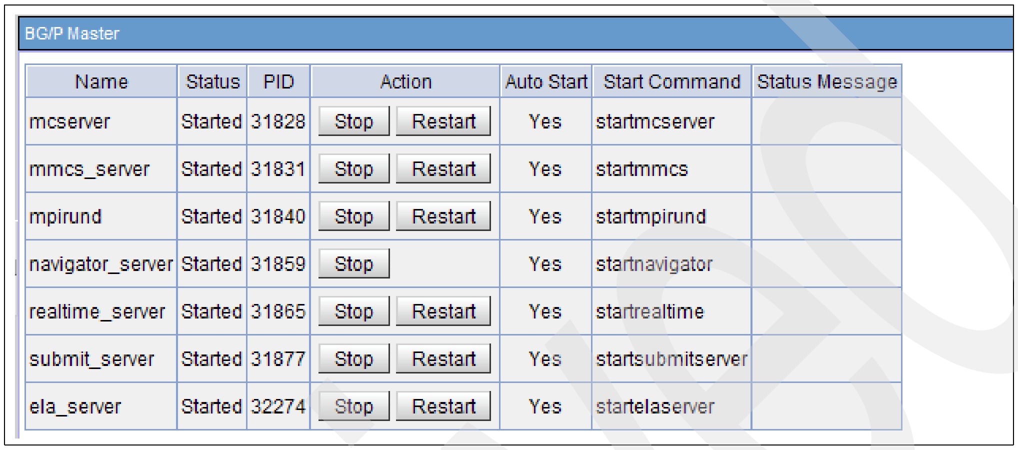
Table 3-3 provides an additional list of options that are used when the `start` and `restart` parameters are used with the **`bgpmaster`** command. These options are ignored if they are used with the `stop`, `reload`, or `status` parameters.

Table 3-3 Start and restart options

Option	Syntax	Description
`--binpath`	*`<directory>`*	Specifies the directory that contains the BGPMaster program. The default is `/bgsys/drivers/ppcfloor/bin`.
`--configpath`	*`<directory>`*	Specifies the path to the directory that contains the configuration file. The default is `/bgsys/local/etc`.
`--useDatabase`	*`<machine serial number>`*	Specifies the machine serial number for the system that is managed by the control system. In most cases, this value will be `BGP`. If a value is not specified, the default value is the system name in the `db.properties` file. Used to separate log files for different Blue Gene systems.
`--db2profile`	*`<db2profile file name>`*	Specifies the db2profile to source for connecting to the database. Default is `db2profile`.
`--dbproperties`	*`<dbproperties file name>`*	Specifies a dbproperties to source for connecting to the database. Default is `db.properties`.
`--baselogdir`	*`<directory>`*	Names the path to the directory where log files are created. The default **`bgpmaster`** log file is `/baselogdir/machine SN /<hostname>-bgpmaster-<date:time>.log`.
`--loglink`	*`<symlink name>`*	Specifies the name of the symlink to the latest (current) log file. The default is `/baselogdir/machine SN /<hostname>-bgpmaster-current.log`.
`--config`	*`<configuration file name>`*	Specifies the name of the **`bgpmaster`** configuration file. The default is `bgpmaster.init`.

3.1.3 The BGPMaster Navigator interface

As shown in Figure 3-1, you can stop, start, and restart each of the control system daemons (with the exception of the navigator_server) from the Navigator interface. When you use the Navigator interface to start or restart a component, the daemon runs under the profile that originally started BGPMaster. You must start or stop BGPMaster itself from the command line.


BG/P Master

Name	Status	PID	Action	Auto Start	Start Command	Status Message
mcserver	Started	31828	Stop Restart	Yes	startmcserver	
mmcs_server	Started	31831	Stop Restart	Yes	startmmcs	
mpirund	Started	31840	Stop Restart	Yes	startmpirund	
navigator_server	Started	31859	Stop	Yes	startnavigator	
realtime_server	Started	31865	Stop Restart	Yes	startrealtime	
submit_server	Started	31877	Stop Restart	Yes	startsubmitserver	
ela_server	Started	32274	Stop Restart	Yes	startelaserver	

Figure 3-1 BGPMaster's Navigator interface

3.2 mcServer

Important: The MMCS server relies on the mcServer to monitor the status of the hardware. If the mcServer is stopped for an extended period, the mmcs_server will fail. In the default configuration, BGPMaster tries to restart both servers if the **`stop`** command has not been issued.

mcServer is a wrapper around the Machine Controller (mc). Machine Controller is a library of functions that provide low level control system access to the hardware. mcServer wraps this library to provide a server and ports to which to connect. mcServer handles arbitrations and controls access to the hardware to prevent simultaneous conflicting operations. It also provides Target Sets, which give the client a way to refer to a collection of hardware with a single name or handle. mcServer also provides some event handling. Clients can register to receive RAS and console events for a particular location or for all events.

Evolution Note: In Blue Gene/L, this low-level control was provided by the idoproxy.

Figure 3-2 illustrates how mcServer provides a layer of arbitration between the hardware and higher level clients. Examples of clients are MMCS, diagnostics, or service actions.

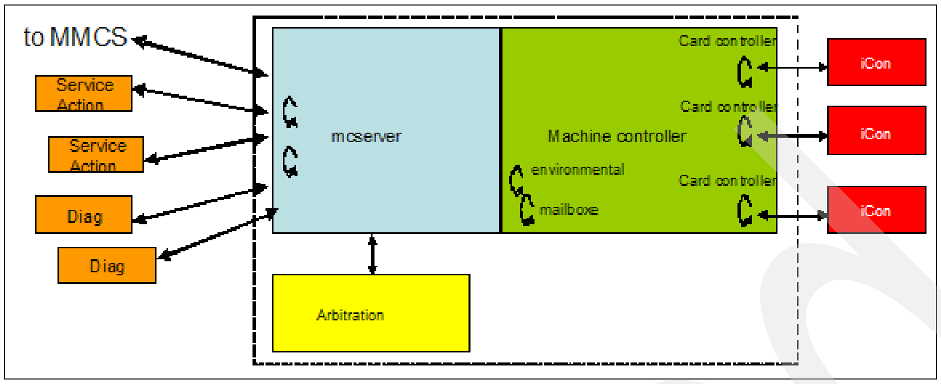


Figure 3-2 mcServer layer

3.2.1 Starting mcServer

Under normal circumstances, BGPMaster starts mcServer. However, there can be occasions that might require a manual start. In these circumstances, you can change to the directory `/bgsys/local/etc`. Verify that the `mcserver_params.xml` file exists and that the data it contains is correct. Example 3-2 shows a sample `mcserver_params.xml` file.

Example 3-2 Sample mcserver_params.xml file

```
<MCServerParams
 clientPort="1206" servletPort="1207"
 machineType="0"
 hostName="localhost"
 autoinit="false" skeletonLocs="false"
 rackRows="1" rackColumns="1">
<Subnet netMask = "255.255.0.0" addressBase = "10.0.100.0" addressCount="255" />
<_hwOptions value="dd2=false" />
<_hwOptions value="debug=2" />
</MCServerParams>
```

Most of the parameters in the `mcserver_params.xml` file are created when the system is installed. For example, `machineType="0"` indicates that it is a standard Blue Gene/P system. It is important to verify that the `rackRows` and `rackColumns` values are correct for your environment and that the default ports of 1206 and 1207 are not already in use.

Table 3-4 provides a list of the debug levels that you can supply as an option when starting the mcServer.

Table 3-4 Debug levels

Debug Level	Explanation
0	Silent mode
1	Default mode, minimal messages
2	Print message for each command received by the server, thread create/terminate
5	Print detailed event messages, events received (delivered to listener) and dropped (no listener)

Debug Level	Explanation
10	Dump XML request/reply objects for most commands
19	Dump XML request/reply objects for all commands, even the ones that return large amounts of data
>19	Detailed debug information concerning mcServer internal operation

After verifying that the file contains the correct information for your environment, issue the following command (assuming that you have at least bgpadmin authority):

```
bgpadmin@servicenode:/bgsys/driver/ppcfloor/bin>./bgpmaster start mcserver
```

Alternatively, if you need to run without BGPMaster you can use this command:

```
bgpadmin@servicenode:/bgsys/local/etc>./mcserver mcserver_params.xml
```

The last option is to start mcServer from the Navigator's BG/P Master page. In the row that begins with `mcserver`, click **Start** in the Action column.

3.2.2 Stopping mcServer

You can stop mcServer using any one of the following methods:

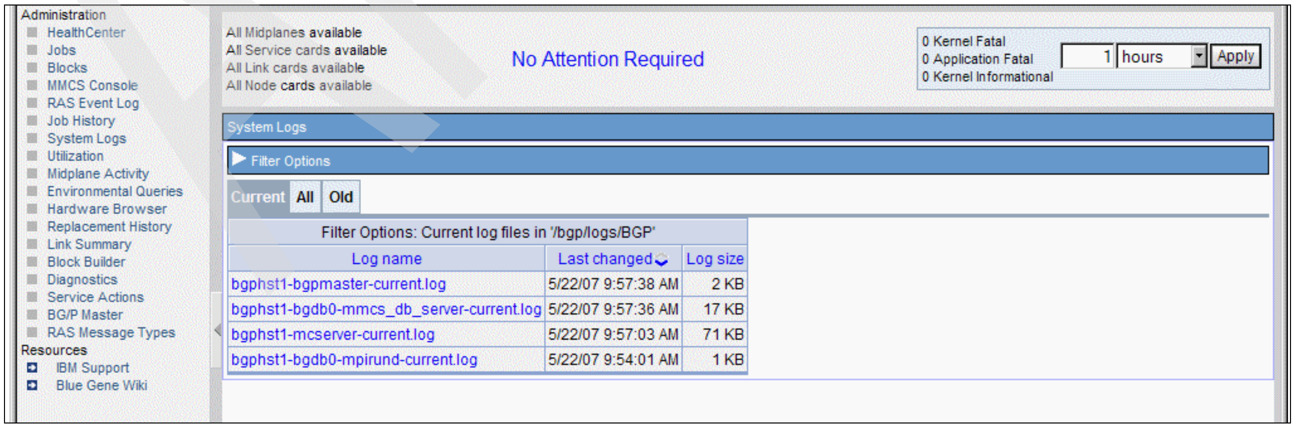
- End it by stopping BGPMaster, which brings down all of the servers.
- End the individual server using the following command in `/bgsys/drivers/ppcfloor/bin`:

  ```
  ./bgpmaster stop mcserver
  ```

- Click **Stop** next to the mcserver entry on the BG/P Master page in the Navigator.

3.2.3 Logging

mcServer writes its logs to the `/bgsys/logs/BGP` directory. The current server instance log file is linked to *xxxxx*`-mcserver-current.log` (where *xxxxx* is the host name of your service node). A new log is created each time mcServer is started. Historical log files are named using the following syntax *xxxxx*`-mcserver-`*yyyy*`-`*mmdd*`-`*hh*`:`*mm*`:`*ss*`.log`. You can also access these logs through the Navigator by clicking **System Logs** in the navigation pane (refer to Figure 3-3). You can select current or historical logs from this interface by clicking the Current, All, or Old tab.



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

No Attention Required

0 Kernel Fatal
0 Application Fatal
0 Kernel Informational

Filter Options: Current log files in '/bgp/logs/BGP'		
Log name	Last changed	Log size
bgphst1-bgpmaster-current.log	5/22/07 9:57:38 AM	2 KB
bgphst1-bgdb0-mmcs_db_server-current.log	5/22/07 9:57:36 AM	17 KB
bgphst1-mcserver-current.log	5/22/07 9:57:03 AM	71 KB
bgphst1-bgdb0-mpirund-current.log	5/22/07 9:54:01 AM	1 KB

Figure 3-3 System Logs

3.3 Midplane Management Control System

You can use the Midplane Management Control System (MMCS) to configure and allocate blocks of Compute and I/O nodes and to run programs on the Blue Gene/P system. Referred to as the *high-level control system*, MMCS runs on the service node and provides an interface to the hardware level system components.

You can access MMCS directly using the MMCS console, Blue Gene Navigator, or through the various scheduler programs (using the scheduler APIs). The MMCS processes that run on the service node also provide the polling services for the Environmental Monitor. By default the Environmental Monitor is active any time the MMCS server is running. For more specific information about MMCS, see Chapter 5, “Midplane Management Control System” on page 115.

3.3.1 Starting the MMCS server

Normally, the BGPMaster processes start or restart the MMCS server when necessary. As with the mcServer processes, there might be an occasion where you need to start the MMCS server manually. In these situations, however, keep in mind that the MMCS server relies on mcServer to communicate with the hardware. Verify that the mcServer is running before starting the MMCS server.

The syntax to start the MMCS server using BGPMaster is:

```
bgpadmin@servicenode:/bgsys/driver/ppcfloor/bin>./bgpmaster start mmcs_server
```

You can also use the **`mmcs_db_server`** command in the `/bgsys/driver/ppcfloor/bin` directory. The drawback to starting the server using this method is that no log files are created.

Alternatively, you can start the MMCS server from the Navigator. Click **BGPMaster** in the navigation pane. Then, click the **Start** or **Restart** button that is associated with the MMCS server (mmcs_server).

3.3.2 Stopping the MMCS server

You can use several methods to stop the MMCS server:

- End it by stopping BGPMaster, which brings down all of the servers that were started when the BGPMaster process was initialized. Use the following command:

  ```
  bgpadmin@servicenode:/bgsys/driver/ppcfloor/bin>./bgpmaster stop
  ```

- Specifically end the MMCS server through BGPMaster using the following command in `/bgsys/drivers/ppcfloor/bin`:

  ```
  ./bgpmaster stop mmcs_server
  ```

- Click the **Stop** button next to the mmcs_server entry on the BG/P Master page in the Navigator.

3.3.3 Logging

Starting the MMCS server through BGPMaster or using the **`startmmcs`** command creates a log file using the following syntax:

`systemname-mmcs_db_server-yyyy-mmdd-hh:mm:ss.log`

It also creates `systemname-bgdb0-mmcs_db_server-current.log` as a symlink to the log file. The link is updated to link to the new file each time the server is started. If you use the **`mmcs_db_server`** command to start the MMCS server, no log file is generated.

You can view log files from the command line or with the Navigator. The MMCS server logs are stored in the `/bgsys/logs/BGP` directory.

3.4 The mpirun daemon

The mpirun daemon is a general purpose job launcher that operates with a client/server relationship. Clients, either on the Blue Gene/P front end node or the service node, submit requests to the server running on the service node. The server process running on the service node is the mpirun daemon (mpirund), which listens for these incoming mpirun requests on port 9874.

Figure 3-4 depicts how mpirun interacts with the remainder of the control system.

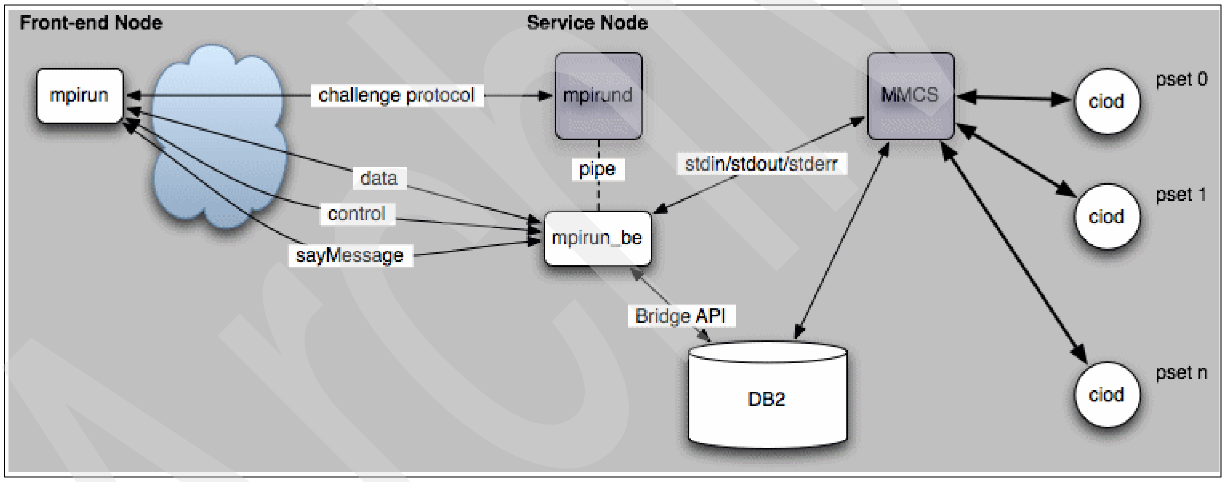


Figure 3-4 The mpirun daemon

Challenge protocol

A *challenge* or *response* protocol is used to authenticate the mpirun client when connecting to the mpirun daemon on the service node. This protocol uses a shared secret to create a hash of a random number to verify that the mpirun front end node has access to the secret. To protect the secret, it is stored in a configuration file that is accessible only by the bgpadmin user on the service node and by a special mpirun user on the front end node. The front-end mpirun binary has its setuid flag enabled so that it can change its UID to match the mpirun user and to read the config file to access the secret.

Front-end and back-end mpirun

The mpirun daemon is comprised of two programs:

- The front-end mpirun program, which is called *mpirun*. Users only come into contact with mpirun on the front end nodes.
- The back-end mpirun program, referred to as *mpirun_be*.

The mpirun daemon can run on any machine in the Blue Gene/P environment. There is no prerequisite for a database client to be installed on these systems, because the front-end program does not access the database. However, mpirun_be does access the database, so the database client must be installed on the service node.

When the authentication is complete, the transaction is handed off to the mpirun_be process. The mpirun_be process initiates a callback to the client based on the information that is passed from the daemon. The client and mpirun_be negotiate the opening of three more connections to complete the mpirun client request.

Bridge APIs

The mpirun_be updates the Blue Gene/P database with information about the job using the Bridge APIs. For more information about this topic, see 1.2.13, "Bridge APIs" on page 13.

MMCS server

The MMCS server monitors the database for updates to various tables (block and job tables). For more information, see 3.3, "Midplane Management Control System" on page 88.

3.4.1 Configuring mpirun

The mpirun daemon uses the following files:

`db.properties`	Contains information about the Blue Gene/P database.
`bridge.config`	Provides the default locations for the I/O node and compute node images used when allocating blocks.
`mpirun.cfg`	Contains the shared secret that is used for the challenge authentication between mpirun and mpirund.

The user that starts BGPMaster must have read access to all of the files.

The db.properties file

The mpirun_be process uses Bridge APIs to make updates to the Blue Gene/P database concerning the mpirun requests that are received. The `db.properties` file contains the information that is required for mpirun to access the database. By default mpirun looks in the `/bgsys/local/etc` directory on the service node for the file when the daemon is starting. The `db.properties` file is referenced in the DB_PROPERTY environment variable by mpirun clients.

Tip: There is a template of the `db.properties` file in the `/bgsys/drivers/ppcfloor/bin` directory (named `db.properties.tpl`). If you do not see the file in the `/bgsys/local/etc` directory, you can copy the template into the directory. When you the copy the file, rename it `db.properties` (that is, remove the `.tpl` extension).

Example 3-3 shows the section of the `db.properties` file that the mpirun daemon uses.

Example 3-3 The db.properties.tpl file

```
database_name=bgdb0
database_user=bgpsysdb
database_password=db24bgp
database_schema_name=bgpsysdb
system=BGP
min_pool_connections=1
mmcs_envs_purge_months=3
mmcs_boot_options=

# DB Polling Governor configuration
blockgovernor_max_concurrent=0
blockgovernor_interval=10
blockgovernor_max_tran_rate=5
blockgovernor_repeat_interval=15
jobgovernor_max_concurrent=0
jobgovernor_interval=10
jobgovernor_max_tran_rate=5
jobgovernor_repeat_interval=15

# Diagnostics variables
DIAGS_PATH=/bgsys/drivers/ppcfloor/diags

# Real-time configuration
realtime_port=32061
realtime_workers=0
realtime_submit_server_port=10247
```

The bridge.config file

The `bridge.config` file contains the path to the images that are used for the I/O nodes and compute nodes to boot. Example 3-4 contains an example of the default file. This file must exist on the service node in the `/bgsys/local/etc` directory. The `bridge.config` file is referenced by the `BRIDGE_CONFIG_FILE` environment variable by mpirun clients.

The mpirun daemon uses the `BGP_DEFAULT_CWD` entry to determine the job's default **`cwd`** if the user fails to provide the **`-cwd`** argument. There is a special case for `BGP_DEFAULT_CWD=$PWD` when mpirun expands that variable to the **`cwd`** where mpirun was launched. As an example, a system administrator might want to change this variable to something similar to `BGP_DEFAULT_CWD=/gpfs` to force jobs without an explicit **`cwd`** to run in their GPFS™.

Example 3-4 The bridge.config file

```
BGP_MACHINE_SN      BGP
BGP_MLOADER_IMAGE   /bgsys/drivers/ppcfloor/boot/uloader
BGP_CNLOAD_IMAGE    /bgsys/drivers/ppcfloor/boot/cns,/bgsys/drivers/ppcfloor/boot/cnk
BGP_IOLOAD_IMAGE
/bgsys/drivers/ppcfloor/boot/cns,/bgsys/drivers/ppcfloor/boot/linux,/bgsys/drivers/ppcfloor/boot
/ramdisk
BGP_LINUX_MLOADER_IMAGE  /bgsys/drivers/ppcfloor/boot/uloader
BGP_LINUX_CNLOAD_IMAGE
/bgsys/drivers/ppcfloor/boot/cns,/bgsys/drivers/ppcfloor/boot/linux,/bgsys/drivers/ppcfloor/boot
/ramdisk
```

```
BGP_LINUX_IOLOAD_IMAGE
/bgsys/drivers/ppcfloor/boot/cns,/bgsys/drivers/ppcfloor/boot/linux,/bgsys/drivers/ppcfloor/boot
/ramdisk
BGP_BOOT_OPTIONS
BGP_DEFAULT_CWD    $PWD
BGP_ENFORCE_DRAIN
```

The mpirun.cfg file

The `mpirun.cfg` file contains the shared secret that is used by the mpirun daemon in the authentication process. Example 3-5 shows the format of the `mpirun.cfg` file. This file needs to exist on the service node and any front end nodes that submit jobs using the mpirun command. Regardless of the system, the files need to match exactly for the authentication process to work. The default location of the `mpirun.cfg` file is in the /bgsys/local/etc/ directory. For security reasons, you can move the file to the `/etc` directory. Only bgpadmin and the special mpirun user need access to the file.

Example 3-5 The mpirun.cfg file

```
CHALLENGE_PASSWORD=BGPmpirunPasswd
```

Tip: If you need a copy of the `mpirun.cfg` file, there is a template in the `/bgsys/drivers/ppcfloor/bin` directory.

3.4.2 Environment variables

The mpirun_be process uses the following environment variables:

- DB_PROPERTIES

  If the DB_PROPERTIES variable is not set, mpirun_be looks for a `db.properties` file in the `/bgsys/local/etc` directory.

- BRIDGE_CONFIG_FILE

  If the BRIDGE_CONFIG_FILE variable is not set, mpirun_be stops.

Both environment variables are validated by mpirund upon startup, because they are always passed to mpirund through the startmpirund script. If either of these environment variables does not exist, mpirund will fail to start.

3.4.3 The front end node

There must be an `mpirun.cfg` file configured on the front end node in the `/bgsys/local/etc` or the `/etc` directory. This file contains the password that is used to authenticate the initial mpirun communications, and access should be restricted to bgpadmin and the mpirun user. As mentioned previously, the contents of this file must match the contents of the file on the service node *exactly*. If the file is moved to the `/etc` directory, only bgpadmin and the special mpirun user should have access to the file for security reasons.

The service node must also be able to resolve the host name of the front end node either through its own host table or from a DNS server. When the initial connection is made, the front end node provides the service node with its host name. In turn, the service node uses the host name in its callback process to open new connections back to the front end node.

3.4.4 Starting the mpirun daemon

By default, the mpirun daemon starts automatically when you start BGPMaster. Starting the server this way allows the BGPMaster process to monitor the server and to restart the daemon if it should fail.

If the mpirun daemon has stopped for some reason, you can start it manually using one of the following methods:

- In the `/bgsys/drivers/ppcfloor/bin` directory, you can issue the following command:

  ```
  bgpadmin@servicenode:/bgsys/driver/ppcfloor/bin>./bgpmaster start mpirund
  ```

- From the Navigator, on the BG/P Master page, you can click the **Start** (or **Restart**) button that is associated with mpirund.

3.4.5 Stopping the mpirun daemon

You can stop the mpirun daemon using one of the following methods:

- End it by stopping BGPMaster, which brings down all of the servers, using the following command:

  ```
  bgpadmin@servicenode:/bgsys/driver/ppcfloor/bin>./bgpmaster stop
  ```

- End the individual server through BGPMaster using the following command in `/bgsys/drivers/ppcfloor/bin`:

  ```
  ./bgpmaster stop mpirund
  ```

- Click **Stop** next to the mpirund entry on the BG/P Master page in the Navigator.

3.4.6 Logging

The default location for the mpirun daemon logs is /bgsys/logs/BGP. The naming syntax used when creating the log files is `<`*`hostname`*`>-mpirund-`*`yyyy-mmdd-hh:mm:ss`*`.log`. When the server is started, a link to the current log (in the same directory) is created. This link file is named `<`*`hostname`*`>-bgdb0-mpirund-current.log`. You can monitor this current file from the command line or Navigator.

3.5 Navigator server

Blue Gene Navigator is a full featured Web-based administration tool. You can also use it to monitor jobs, blocks, current usage, and the availability of the Blue Gene hardware. The navigator_server process powers the Navigator Web pages. When the server is running, you can access the secure Web pages at:

```
https://mybluegenep:32072/BlueGeneNavigator
```

You do need to perform some configuration steps to set up the Web pages. For more information about setup, see Chapter 2, “Blue Gene Navigator” on page 17.

3.5.1 Starting the navigator_server process

The Blue Gene Navigator (navigator_server) is set to start automatically when BGPMaster is started. In the event that you need to start the server manually and that you want BGPMaster to monitor it, use the following command:

```
$ /bgsys/driver/ppcfloor/bin/bgpmaster start navigator_server
```

3.5.2 Stopping the navigator_server

If the Navigator was started with BGPMaster, it ends if you stop the BGPMaster process. You can also end the Navigator server individually by issuing the following command:

```
$ /bgsys/driver/ppcfloor/bin/bgpmaster stop navigator_server
```

You can stop the navigator_server process from the Navigator itself by clicking the **Stop** button in the row that is associated with the navigator_server. At that point, you have to restart the server from the command line.

3.5.3 Logging

The Navigator writes logs to the `/bgsys/logs/BGP` directory. The file `<hostname>-bgdb0-navigator-current.log` is a link to the log of the Navigator instance that is currently running. The actual files to which the logs are written use the following naming syntax:

```
<hostname>-navigator-yyyy-mmdd-hh:mm:ss.log
```

You can view or tail the files from the command line. You can also view or monitor the logs from the Navigator by clicking **System Logs** in the navigation pane.

3.6 Real-time server

The Real-time server provides real-time clients, such as schedulers, with notifications about events to which they have subscribed. These events cause updates in the database, for example, when a partition is freed. Prior to V1R3M0, triggers in the database updated the state tracked by the Real-time server. In turn, the server sent the updates to the Real-time client based on the clients' callback requests.

When a scheduler is initially started, the state of the system is acquired from the database using the Bridge APIs (`rm_get_*`). This information provides the scheduler with a snapshot to use as a starting point. As subsequent events occur, the scheduler can use the Real-time server to keep updated with the changes. For more detailed information, refer to *IBM System Blue Gene Solution: Blue Gene/P Application Development*, SG24-7287.

With V1R3M0, there is a significant change to the way the Real-time server works. The server now runs as a separate process, waiting for connections from registered clients. When a client connects to the server, the server begins to monitor the DB2 transaction log for updates to the database. Clients can register for updates about job, partition, and hardware state changes or any combination of the three. The server filters the information and sends only the requested data to the clients.

Evolution Note: In Blue Gene/L, when schedulers wanted an update, they had to poll the database using the Bridge APIs. This constant polling can have a negative impact on performance.

Blue Gene/P database tables of interest to the schedulers have a sequence identification (seqid) field in them. As events enter the database, the seqid increments. The event with largest value in the seqid field is always the most recent event in that table.

The following database tables contain the sequence identification field:

TBGPBLOCK	blocks
TBGPJOB	jobs
TBGPMIDPLANE	base partitions
TBGPNODECARD	node cards
TBGPSWITCH	switches
TBGPCABLE	cables

3.6.1 Configuring the Real-time server

In order for the Real-time server to read the transaction logs, you need to configure the database to be recoverable (a change from releases prior to V1R3M0). To configure the database for the Real-time server, follow these steps:

1. Switch users to *bgpsysdb*, and update the database to make it recoverable using the following commands:

   ```
   su - bgpsysdb
   db2 "update database configuration using logarchmeth1 DISK:/dbhome/bgpsysdb/dblogs/bgdb0/"
   ```

2. Next, stop and restart DB2 so the changes take effect:

   ```
   db2 force application all
   db2stop
   db2start
   ```

3. After the database restarts, you must make a backup:

   ```
   db2 "BACKUP DATABASE BGDB0"
   ```

4. Exit the bgpsysdb profile (to the bgpadmin profile) to continue configuring the Real-time server:

   ```
   exit
   ```

You set up the Real-time server by running the `configureRealtime.pl` program. Running the program configures the database schema for the Real-time server code. You need to run the program prior to starting the Real-time server. The server runs using the *bgpsysdb* db2 profile. You need to verify that this user can read the db.properties file and can write to the log directory (`/bgsys/logs/BGP` unless otherwise specified). This program has the following modes of operation:

`install`	Adds any stored procedure that is not defined or alters any stored procedure that is defined, and then adds any trigger that is not defined.
`update`	Performs the same action as `install` if any stored procedure or trigger is found. Otherwise, does nothing.
`uninstall`	Removes all stored procedures and triggers.

The default mode is `update`.

For a new installation, you typically run the following sequence of commands:

```
$ . ~bgpsysdb/sqllib/db2profile
$ cd /bgsys/drivers/ppcfloor/schema
$ ../bin/logOutput --svcname configureRealtime ./configureRealtime.pl install
```

The following additional options are available for configureRealtime.pl:

--configpath	Overrides the default configuration path of `/bgsys/local/etc`.
--dbproperties	Overrides the default `db.properties` file of `<`*`configpath`*`>/db.properties`.
--properties	Overrides the default configureRealtime.properties files of `<`*`exepath`*`>/configureRealtime.properties`.
--help	Prints help text and exits.

Attention: Beginning in V1R3M0, the Real-time server runs as a separate program rather than inside the DB2 server as it did in prior releases. You can find the steps to update the configuration in the V1R3M0 readme file.

3.6.2 Starting the realtime_server process

Beginning in V1R3M0, the `bgpmaster.init` starts the Real-time server automatically. In releases prior to V1R3M0, the default does not start the server automatically. In the earlier releases, you have to modify the `bgpmaster.init` file to set the server to start with BGPMaster. To autostart the Real-time server, uncomment the following line in the `.init` file:

```
#start   realtime_server
```

To start the Real-time server manually, run one of the following commands:

- `$ /bgsys/driver/ppcfloor/bin/bgpmaster start realtime_server`
- `$ /bgsys/local/etc/startrealtime`

You can also start the Real-time server through the Blue Gene Navigator. In the navigation pane, click the BG/P Master link. In the main content area, click **Start** (to the right of the realtime_server).

3.6.3 Stopping the Real-time server

You can stop the Real-time server using one of the following methods:

- End all of the BGPMaster monitored servers (and BGPMaster) by issuing the following command:

  ```
  bgpadmin@servicenode:/bgsys/driver/ppcfloor/bin>./bgpmaster stop
  ```

- Stop just the Real-time server with the following command:

  ```
  bgpadmin@servicenode:/bgsys/driver/ppcfloor/bin>./bgpmaster stop
  realtime_server
  ```

- Click the **Stop** button that is associated with the realtime_server on the BG/P Master page in Navigator.

3.6.4 Log cleanup

You might want to create a cron job that runs daily to clean up archived logs. Example 3-6 shows a sample script named `/etc/cron.daily/bluegene-db-log-cleaner.sh` that we created to do the cleanup.

Example 3-6 Sample clean up script

```
#!/bin/bash
find /dbhome/bgpsysdb/dblogs/bgdb0/ -type f -mtime +1 -exec rm {} \;
```

Make sure that the new file is executable by running the following command:

```
chmod +x /etc/cron.daily/bluegene-db-log-cleaner.sh
```

3.7 Event Log Analysis

The Event Log Analysis (ELA) server generates alerts for RAS events that require administrator attention. ELA is composed of a pipeline of components for event monitoring, event analyzing, alert filtering, and alert listening. See Chapter 8, “Event Log Analysis” on page 155 page for a complete description of ELA.

3.7.1 Configuring ELA

Restriction: ELA requires that Python and PyDB2 be installed on the service node. The instructions for setting up this environment are in Appendix F, “Setting up the Event Log Analysis Environment” on page 269

The ELA server uses the following configuration files:

- `ela_config.xml`
- `db.properties`
- RAS filter

The ela_config.xml file

This file is the main configuration file for ELA. It contains configuration settings that are used to determine which components are activated during an ELA start. There is a sample file for ELA configuration file located at `/bgsys/drivers/ppcfloor/bin/ela_config.xml`. While logged in as bgpadmin, copy this file to the local configuration using the following command:

```
cp /bgsys/drivers/ppcfloor/bin/ela_config.xml /bgsys/local/etc/ela_config.xml
```

After the file is copied, verify that it has the correct file authorities and ownership. Make sure the owner and group are set to `bgpadmin` with read/write authority. No one else should have access to the file. If needed, use the following commands to set the ownership and authorities on the file:

```
chown bgpadmin:bgpadmin /bgsys/local/etc/ela_config.xml
chmod 660 /bgsys/local/etc/ela_config.xml
```

Example 3-7 shows a sample of an `ela_config.xml` file.

Example 3-7 ela_config.xml

```
<ela>
    <config_property name="hostname" value="bgpsn"/>

    <!-- The database modules provide connections to a database. -->
    <databases>
        <database module_name="database.Connection" enabled="true"/>
    </databases>

    <!-- The event monitors notify ELA of RAS events.  -->
    <!-- Only one event monitor can be enabled.        -->
    <event_monitors>
        <monitor module_name="event_monitor.PeriodicRASEventQuery"
enabled="false">
       <config_property name="period" value="60"/>
   </monitor>
        <monitor module_name="event_monitor.HistoricRASEventQuery"
enabled="false">
       <config_property name="startTime" value="2009-02-03-20.28.08.0"/>
       <config_property name="stopTime"  value="2009-02-04-17.28.09.0"/>
   </monitor>
        <monitor module_name="event_monitor.RealtimeEventMonitor" enabled="true">
   </monitor>
    </event_monitors>

    <!-- The analyzers perform analysis and generate alerts.  -->
    <!-- All can be enabled.                                  -->
    <analyzers>
        <analyzer module_name="analyzer.HardwareInErrorAnalyzer" enabled="true"/>
        <analyzer module_name="analyzer.JobFatalRasAnalyzer" enabled="true"/>
        <analyzer module_name="analyzer.ThresholdExceededAnalyzer" enabled="true">
       <config_property name="excludeMsgIDs" value="KERN_1014,KERN_1015"/>
   </analyzer>
        <analyzer module_name="analyzer.CommonModeAnalyzer" enabled="true">
            <config_property name="windowTime" value="1 MINUTE"/>
        </analyzer>
        <analyzer module_name="analyzer.SampleAnalyzer" enabled="false"/>
    </analyzers>

    <!-- The alert filters can veto an alert.                 -->
    <!-- All can be enabled.                                  -->
    <alert_filters>
        <filter module_name="alert_filter.DuplicateFilter" enabled="true"/>
        <filter module_name="alert_filter.ActiveAlertCeilingFilter"
enabled="false">
       <config_property name="ceiling" value="100"/>
   </filter>
        <filter module_name="alert_filter.CommonModeFilter" enabled="true"/>
    </alert_filters>


    <!-- The alert listeners process alerts.                  -->
    <!-- All can be enabled.                                  -->
    <alert_listeners>
        <listener module_name="alert_listener.DatabaseLogger" enabled="true"/>
        <listener module_name="alert_listener.EmailSender" enabled="false">
       <config_property name="smtp_server" value="localhost"/>
       <email_from  name="ELA Server" userid="ela_server@abc.com"/>
       <email_to name="SysAdmin1" userid="admin1@abc.com"/>
       <email_to name="SysAdmin2" userid="admin2@abc.com"/>
   </listener>
        <listener module_name="alert_listener.SampleListener" enabled="false">
       <config_property name="verbose" value="true"/>
   </listener>
    </alert_listeners>
</ela>
```

The db.properties file

ELA uses information in this file when making a connection to and when accessing the database. By default, ELA looks for the `db.properties` file in the `/bgsys/local/etc` directory when the ELA server is starting.

The RAS filter file

This file is used to change specifications (for example, severity or control action) of RAS events, which are be read and taken into consideration by ELA when processing those RAS events. By default, ELA looks for the `ras_environment_filter.xml` file in the `/bgsys/local/etc` directory when ELA is starting.

3.7.2 Starting the ELA server process

To start the ELA server, you must first copy the `startelaserver.tpl` file into the `/bgsys/local/etc` directory. Then, set the correct authorities to the file using the following commands:

```
cp /bgsys/drivers/ppcfloor/bin/startelaserver.tpl /bgsys/local/etc/startelaserver
chown bgpadmin:bgpadmin /bgsys/local/etc/startelaserver
chmod 770 /bgsys/local/etc/startelaserver
```

By default, the ELA server is not set to start automatically when BGPMaster starts. To set the server to start with BGPMaster, uncomment (remove the # sign) the following line in the `/bgsys/local/etc/bgpmaster.init` file:

```
#start ela_server
```

To start the ELA server manually, run one of the following commands:

- `/bgsys/driver/ppcfloor/bin/bgpmaster start ela_server`
- `/bgsys/local/etc/startelaserver`

You can also start the ELA server from Blue Gene Navigator on the BG/P Master page by clicking the **Start** button next to the ELA server.

3.7.3 Stopping the ELA server process

You can stop the ELA server using any one of the following methods:

- Stop BGPMaster, which stops all of the servers that were started by BGPMaster, using the following command:

  ```
  /bgsys/driver/ppcfloor/bin/bgpmaster stop
  ```

- Stop just the ELA server through BGPMaster using the following command:

  ```
  /bgsys/driver/ppcfloor/bin/bgpmaster stop ela_server
  ```

- Click the **Stop** button that is associated with the ELA server on the BG/P Master page in Blue Gene Navigator in the Action column (refer to Figure 3-1 on page 85).

3.7.4 Logging

The default location for the ELA server log is the `/bgsys/logs/BGP` directory. A new log is created each time ELA server is started. That log file uses the following syntax:

```
SNhostname-ela_server-yyyy-mmdd-hh:mm:ss.log
```

It also creates a link to the current log file named `SNhostname-ela_server-current.log`. The link is updated to the new log file each time the server is started.

You can access these log files from the command line or through the Navigator by clicking **System Logs** in the navigation pane.

4

Blocks

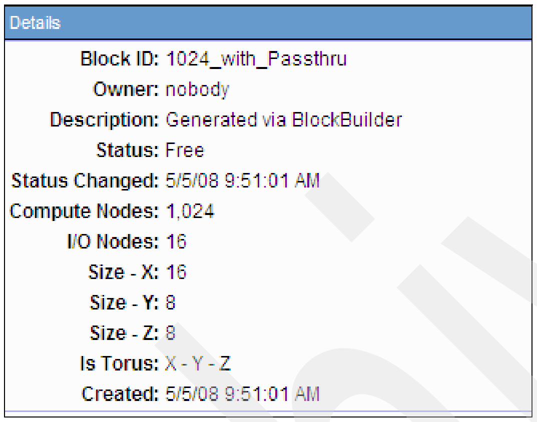
Blocks, or *partitions* as they are also known, provide the means to identify all or part of the hardware on Blue Gene/P onto which jobs can be submitted. In this chapter, we explain blocks in more detail.

4.1 Blocks

Before you can submit any job to the Blue Gene core, you must create in the database a block or partition that defines the resources that are used for the job. The hardware and cabling of the Blue Gene/P core system determines which blocks you can create on your system. Block definitions are used to describe the Blue Gene/P resources that are used for running jobs.

Blocks can be either *dynamic* or *static*. An example of a dynamic block might be one that is created by a job scheduler to run a specific job and then deleted when the job ends. A static block is a partition that is created as a general purpose block and that is left on the system for use as needed. Each block definition is identified with a unique *block ID*.

Figure 4-1 shows how a block definition displays in the Navigator.



Block ID:	1024_with_Passthru
Owner:	nobody
Description:	Generated via BlockBuilder
Status:	Free
Status Changed:	5/5/08 9:51:01 AM
Compute Nodes:	1,024
I/O Nodes:	16
Size - X:	16
Size - Y:	8
Size - Z:	8
Is Torus:	X - Y - Z
Created:	5/5/08 9:51:01 AM

Figure 4-1 Block Details in the Navigator

The Base Partition of any Blue Gene/P system is a *midplane*. Midplanes are cabled together using link cards and data cables to create larger blocks. The size of the blocks that you can create and which midplanes you can use for the blocks are determined by the way your Blue Gene/P core is cabled. You can configure blocks that are created using one or more full midplanes as either a three-dimensional (3D) Mesh or a looping 3D Torus network. Later in 4.2, “Displaying and creating blocks using Blue Gene Navigator” on page 103 and 4.3, “Creating blocks using Midplane Management Control System console” on page 108, we discuss how to create blocks for multiple rack systems.

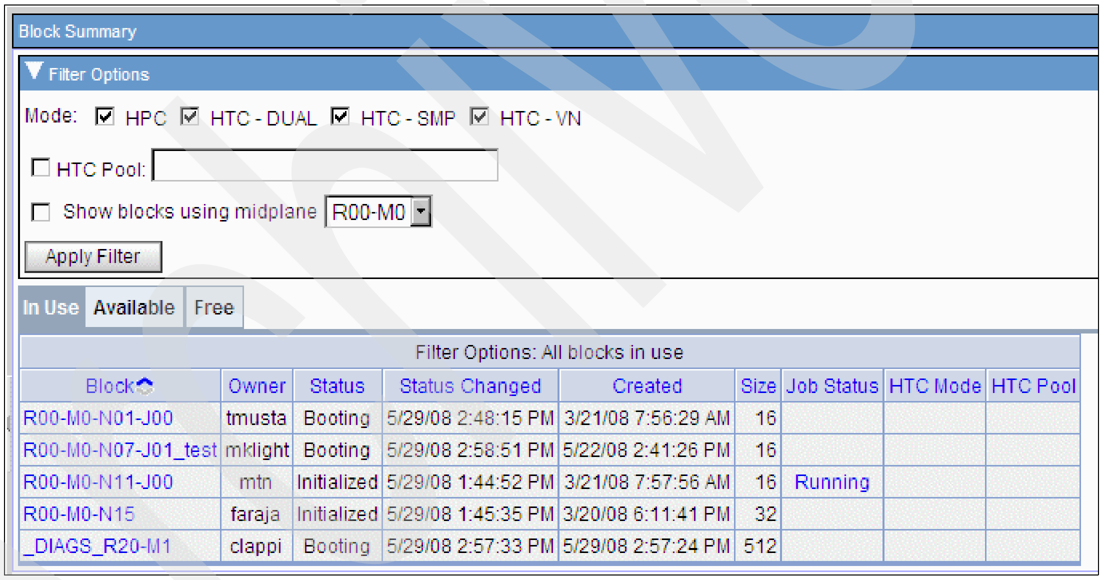
You can subdivide Base Partitions into small blocks. These blocks depend on the network fabric that is provided by the midplane. In addition, there are specific rules about creating these blocks. You can create small blocks sizes, such as 16, 32, 64, 128, or 256 compute nodes. The size of small blocks that you can create depends on the number of I/O nodes that your system has. Each block needs at least one I/O node. So, theoretically, you can subdivide a midplane that is fully populated with I/O nodes into 32 blocks, each with 16 compute nodes and one I/O node. Small blocks can only use mesh data connections, which are actually hardwired into the node cards. As we mentioned previously, only blocks that consist of a single midplane or more can take advantage of the 3D Torus network.

You create dynamic blocks using a job submission program, such as LoadLeveler or mpirun. The user that submits the job tells the submission program details about the Blue Gene/P resources, such as the number of processors that are needed for the job and the size of processor set (the number of I/O nodes) to create for the job, the microloader, compute node images, and I/O node images to use. The job submission program creates the block based on this information using the MMCS Bridge APIs and any of the Blue Gene/P resources that are available at the time to run the job. The user cannot choose which resources are used. When the job has completed, the block is freed and destroyed, and it no longer exists in the MMCS database.

The system administrator creates static blocks, and they remain in the MMCS database to be reused when needed. These block definitions also include the specific hardware resources on the Blue Gene core, the pset size of the block, as well as the microloader, I/O node images, and compute node images. These static blocks allow the system administrator to assign a block for a particular user to use specific core resources. When users submit their jobs, they need only to specify which static block they are going to use to execute the job.

4.2 Displaying and creating blocks using Blue Gene Navigator

In Blue Gene Navigator the *Blocks* and *Block Builder* areas pertain to blocks. Selecting **Blocks** opens the page shown in Figure 4-2. This page is available in both the administrator's and the user's version of the Navigator. You can chose to view blocks that are in use, available, or free. The default is to show the blocks that are available for the entire system (all Blue Gene/P hardware), but you can filter the view (by expanding the Filter Options) to a single midplane by selecting that specific midplane in the Filter Options. You can also filter by the mode or by High Throughput Computing (HTC) pool.



Mode: [✓ HPC] [✓ HTC - DUAL] [✓ HTC - SMP] [✓ HTC - VN]

☐ HTC Pool:

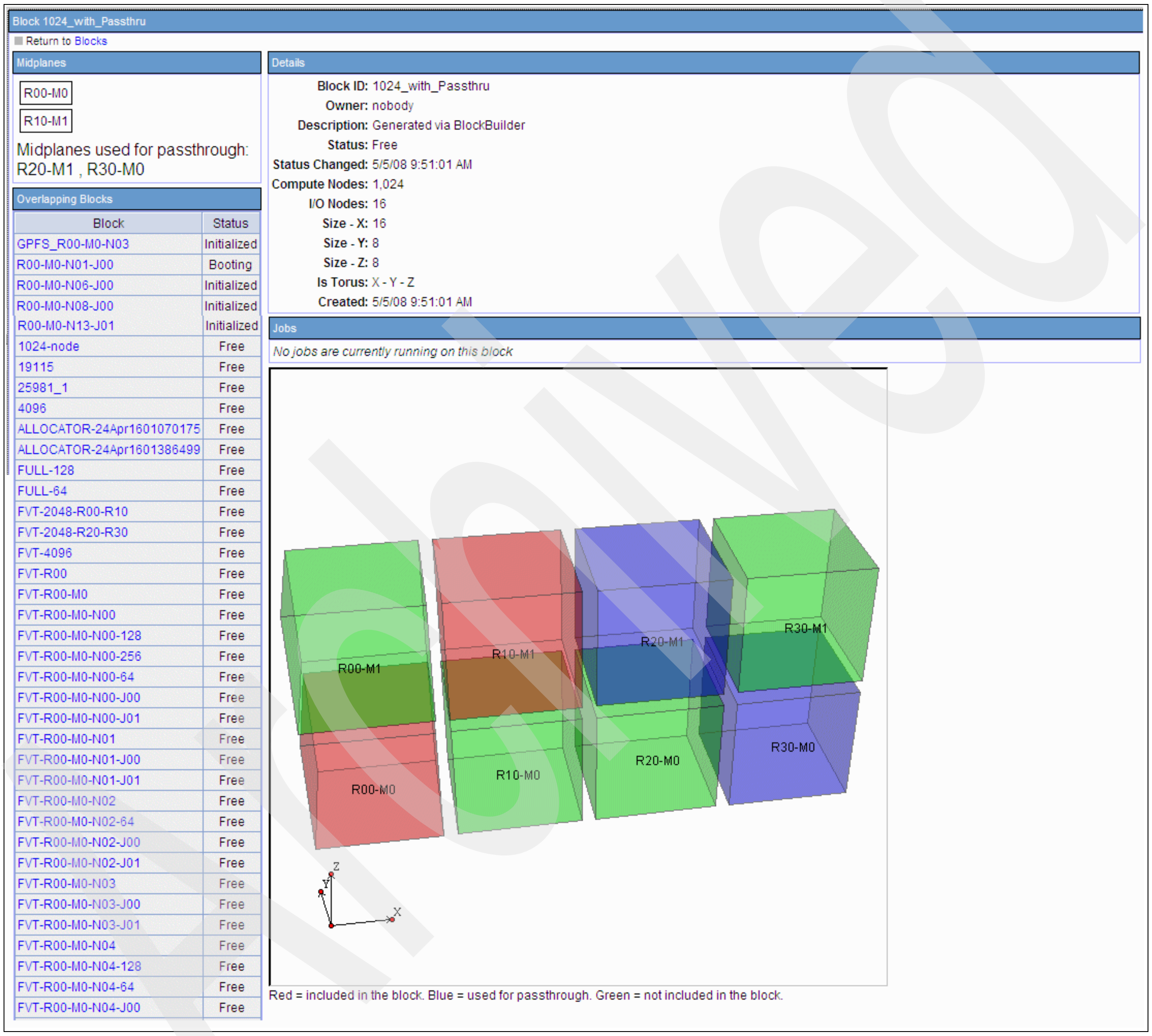
☐ Show blocks using midplane R00-M0

Filter Options: All blocks in use

Block	Owner	Status	Status Changed	Created	Size	Job Status	HTC Mode	HTC Pool
R00-M0-N01-J00	tmusta	Booting	5/29/08 2:48:15 PM	3/21/08 7:56:29 AM	16			
R00-M0-N07-J01_test	mklight	Booting	5/29/08 2:58:51 PM	5/22/08 2:41:26 PM	16			
R00-M0-N11-J00	mtn	Initialized	5/29/08 1:44:52 PM	3/21/08 7:57:56 AM	16	Running		
R00-M0-N15	faraja	Initialized	5/29/08 1:45:35 PM	3/20/08 6:11:41 PM	32			
_DIAGS_R20-M1	clappi	Booting	5/29/08 2:57:33 PM	5/29/08 2:57:24 PM	512			

Figure 4-2 Block Summary page

Notice that the block names are links. Clicking one of the links opens the Block Details page shown in Figure 4-3. This page contains information about the block, such as its size, the number of I/O nodes, and whether it is a torus. Blocks that use all or some of the hardware included in this block are listed on the left side of the page. Blocks that consist of a midplane (512 nodes) or more are also displayed using the Block Visualizer. The Block Visualizer shows a three-dimensional view of the entire system. Midplanes used in a block, pass-through midplanes, and midplanes that are not involved in the block are all colored differently.



Block 1024_with_Passthru

Return to Blocks

Midplanes

R00-M0

R10-M1

Midplanes used for passthrough: R20-M1 , R30-M0

Overlapping Blocks

Block	Status
GPFS_R00-M0-N03	Initialized
R00-M0-N01-J00	Booting
R00-M0-N06-J00	Initialized
R00-M0-N08-J00	Initialized
R00-M0-N13-J01	Initialized
1024-node	Free
19115	Free
25981_1	Free
4096	Free
ALLOCATOR-24Apr1601070175	Free
ALLOCATOR-24Apr1601386499	Free
FULL-128	Free
FULL-64	Free
FVT-2048-R00-R10	Free
FVT-2048-R20-R30	Free
FVT-4096	Free
FVT-R00	Free
FVT-R00-M0	Free
FVT-R00-M0-N00	Free
FVT-R00-M0-N00-128	Free
FVT-R00-M0-N00-256	Free
FVT-R00-M0-N00-64	Free
FVT-R00-M0-N00-J00	Free
FVT-R00-M0-N00-J01	Free
FVT-R00-M0-N01	Free
FVT-R00-M0-N01-J00	Free
FVT-R00-M0-N01-J01	Free
FVT-R00-M0-N02	Free
FVT-R00-M0-N02-64	Free
FVT-R00-M0-N02-J00	Free
FVT-R00-M0-N02-J01	Free
FVT-R00-M0-N03	Free
FVT-R00-M0-N03-J00	Free
FVT-R00-M0-N03-J01	Free
FVT-R00-M0-N04	Free
FVT-R00-M0-N04-128	Free
FVT-R00-M0-N04-64	Free
FVT-R00-M0-N04-J00	Free

Details

Block ID:	1024_with_Passthru
Owner:	nobody
Description:	Generated via BlockBuilder
Status:	Free
Status Changed:	5/5/08 9:51:01 AM
Compute Nodes:	1,024
I/O Nodes:	16
Size - X:	16
Size - Y:	8
Size - Z:	8
Is Torus:	X - Y - Z
Created:	5/5/08 9:51:01 AM

Jobs

No jobs are currently running on this block

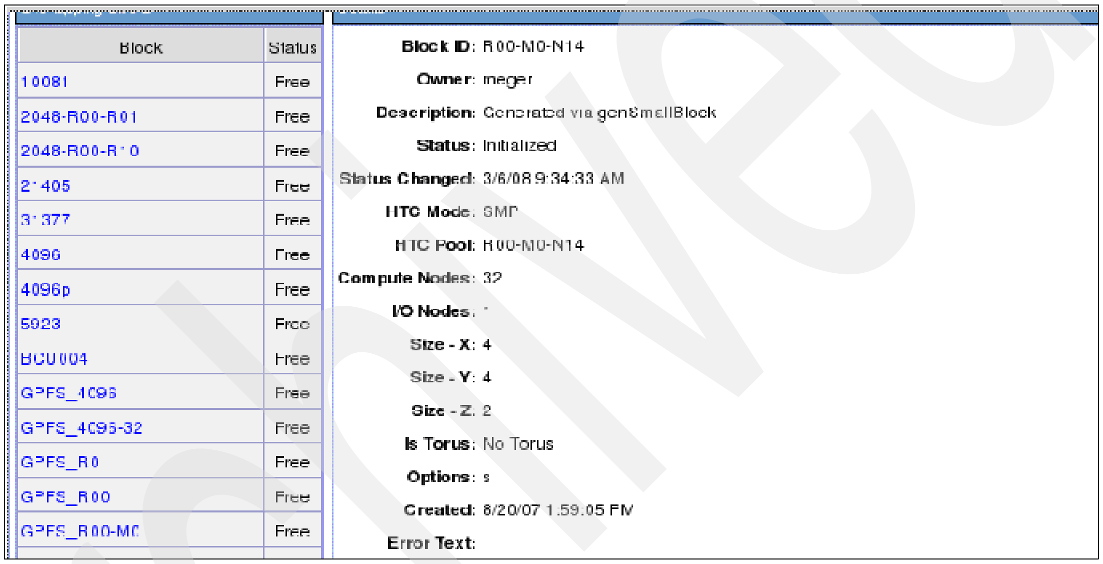
Red = included in the block. Blue = used for passthrough. Green = not included in the block.

Figure 4-3 Block Details page

Figure 4-4 shows the block details of a block that is booted in HTC SMP mode. Notice that in the Jobs section, the displays shows the number of jobs that are currently running and the total number of jobs that are complete. In addition to the usual information returned, you also see sections containing Hardware Used and Failed Nodes.

Blocks booted in `htc=linux` mode are displayed the same as other HTC blocks, except that HTC Mode equals *Linux* rather than SMP, DUAL, or VN, and the *Options* field has an *l* (lower case L). Blocks that are booted in Linux mode have an additional detail, *CN Load Image*. The CN Load Image shows either that the block was booted with the default Linux image or the location of the image that was used.

The Blue Gene/P version of HTC does not try to recover nodes that fail. A single failed node does not cause the entire block to become unavailable as with a block that boots in HPC mode. The failed nodes are removed from the pool of available nodes, and the remaining jobs continue to run.

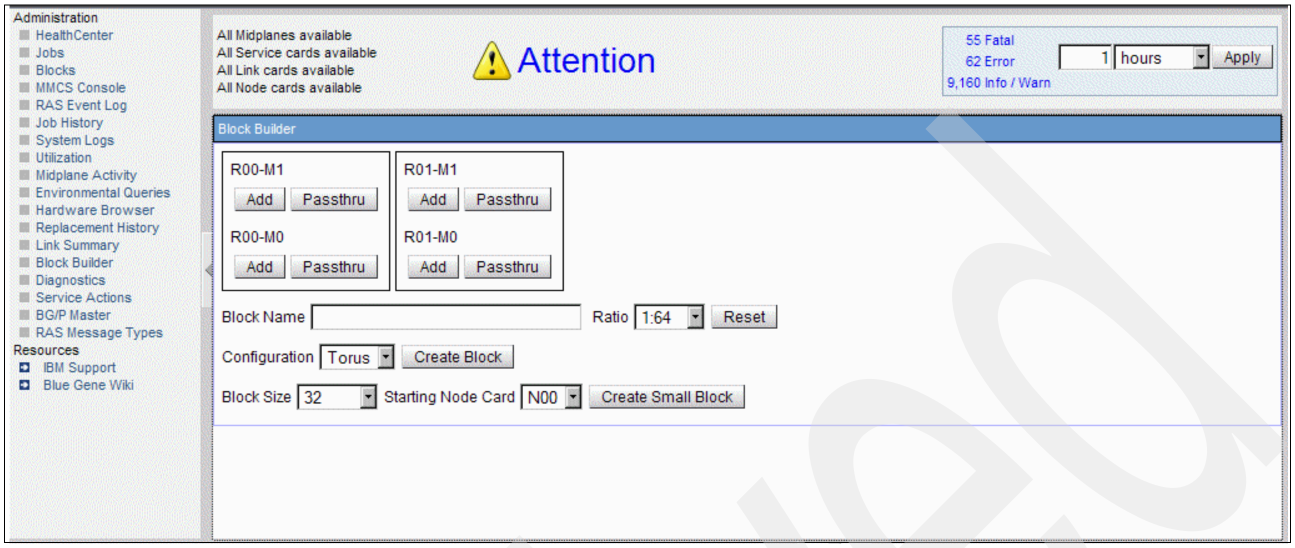


Block	Status
10081	Free
2048-R00-R01	Free
2048-R00-R10	Free
2?405	Free
3?377	Free
4096	Free
4096p	Free
5923	Free
BCU004	Free
GPFS_4096	Free
GPFS_4096-32	Free
GPFS_R0	Free
GPFS_R00	Free
GPFS_R00-M0	Free

Block ID: R00-M0-N14
Owner: meger
Description: Generated via genSmallBlock
Status: Initialized
Status Changed: 3/6/08 9:34:33 AM
HTC Mode: SMP
HTC Pool: R00-M0-N14
Compute Nodes: 32
I/O Nodes: 1
Size - X: 4
Size - Y: 4
Size - Z: 2
Is Torus: No Torus
Options: s
Created: 8/20/07 1:59:05 PM
Error Text:

Figure 4-4 Block details

Figure 4-5 shows the Block Builder page. You can navigate to this page by selecting the Block Builder option in the navigation pane of the Navigator interface.



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

1 hours

Ratio 1:64

Configuration Torus

Block Size 32 Starting Node Card N00

Figure 4-5 Block Builder

Just below the Block Builder title bar there is one or more block diagrams that contain a representation of all the midplanes in your system. Figure 4-6 depicts a system that consists of two racks. To create a block that consists of at least one base partition (midplane), click **Add** for each of the midplanes that you want included in the block. If you need to use a midplane's link card to complete the block's torus, without including that midplane's compute or I/O nodes in the block, click **Passthru** for the corresponding midplane.

Notice that as you choose the midplanes to be added to the block, the midplane label is outlined in red. If you choose to use a midplane as a pass through, the midplane label is outlined in blue. Additionally, after you chose either the Passthru option or add more than one midplane to your block, the small block options (the last row of options in Figure 4-5) are removed from view.

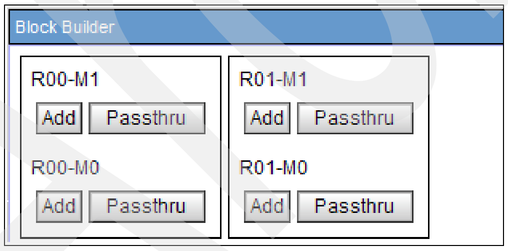


Figure 4-6 Midplane selection

4.2.1 Creating a block of one or more midplanes

To create a block consisting of one or more midplanes using Blue Gene Navigator, follow these steps:

1. Select the midplane (or midplanes) that you want to include in your block.
2. Provide a name for the block. The Block Name can be up to 32 characters in length and must contain only alphanumeric characters, dashes, and underscores.

3. Next, select the ratio. This ratio reflects the number of I/O nodes to compute nodes. You can compute the ratio by dividing the number of compute nodes in the block by the number of I/O nodes that are included.
4. Finally, select the Configuration, which will be either Torus or Mesh, and click **Create Block**.

4.2.2 Creating a small block

Using the Navigator in Blue Gene/P, you can also create small blocks (sub-midplane blocks) from the Web interface. To create a small block, follow these steps:

1. First, select the midplane on which you want to create the block. Click **Add**, which outlines the selected midplane in red.
2. Supply a name for the block. Again, the name can contain up to 32 characters.
3. Next, select the ratio of I/O nodes to compute nodes, which can be determined by the number of compute nodes in the block divided by the number of I/O nodes.

   Attention: Keep in mind that there must be at least one I/O node included in the block. The I/O node (or nodes) also has to be contained in one of the node card (or cards) that hold the compute nodes selected for the block.

   The Configuration for a small block will always be a Mesh.
4. Select the Block Size from the pull-down menu choices:

16-J00	As you face the node card, this block is the I/O node and compute nodes on the left side of the node card.
16-J01	As you face the node card, this block is the I/O node and compute nodes on the right side of the node card.
32	All of the compute nodes and at least one of the I/O nodes on a node card.
64	All of the compute nodes and at least one of the I/O nodes on two node cards. The starting node card must be N0, N2, N4, N6, N8, N10, N12, or N14.
128	All of the compute nodes and at least one I/O node on four node cards. The starting node card must be N0, N4, N8, or N12.
256	All of the compute nodes and at least at least two I/O nodes in eight node cards. The starting node card must be N0 or N8. For example, if you are creating a block with 256 compute nodes and a ratio of 1:128 using the node cards in the front of the rack, your starting node card is N0. The I/O nodes for this ratio will be in the N0 and N4 node cards.

5. Choose the Starting node card from the pull-down menu. For the small blocks that consist of 16 or 32 compute nodes, you can use any node card in the selected midplane, provided that it has the required I/O nodes. For the small blocks that are made up of 64 or more compute nodes, you must use one of the node cards that are listed in the description in the previous step.
6. After you select the starting node card, click **Create Small Block**. A message displays at the bottom of the window confirming that the block was created or notifying you if there was an error.

4.3 Creating blocks using Midplane Management Control System console

The other interface that is used to create static blocks is Midplane Management Control System (MMCS). In this section, we cover only the commands that relate to creating blocks. Chapter 5, "Midplane Management Control System" on page 115 goes into more detail about the remainder of the supported MMCS commands. The supported set of commands for generating blocks are:

- **`genblock`**
- **`genblocks`**
- **`gensmallblock`**
- **`genBPblock`**
- **`genfullblock`**

Attention: Block names (referred to as <*blockid*> in the following sections) can be up to 32 alphanumeric characters, dashes, and underscores.

genblock

The **`genblock`** command is used to generate a block that consists of one base partition. The MMCS command uses the following syntax:

```
genblock <blockid> <midplane> <machineSN> <psetsize>
```

The parameter's definitions are as follows:

<*blockid*>	The name of the block to be created in the BGPBlock table.
<*midplane*>	The POSINMACHINE (for example, `R00-M1`) value from the BGPMidplane table.
<*machineSN*>	The serial number defined in the BGPMACHINE table (usually `BGP`).
<*psetsize*>	The ratio of compute nodes to each I/O node (which depends on the hardware configuration).

The following command creates a block named *Example_1* that consists of a base partition using the hardware resources from R02-M0 (see Figure 4-7). The <*psetsize*> suggests that the rack has only eight I/O nodes per midplane.

```
mmcs$  genblock Example_1 R02-M0 BGP 64
```

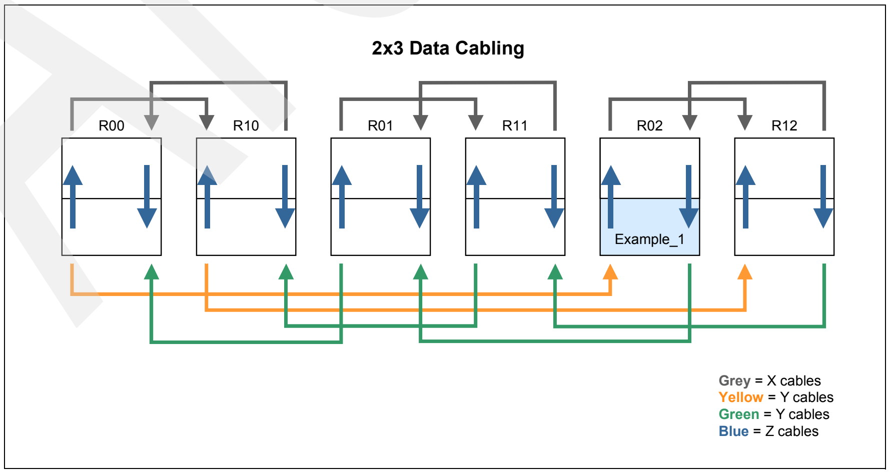


Figure 4-7 The genblock command creates a single Base Partition block

genblocks

The **genblocks** command generates a block for each base partition on the system. The **genblocks** command uses the following syntax:

```
mmcs$ genblocks  <blockidprefix> <machineSN> <psetsize>
```

The parameter's definitions are as follows:

<blockidprefix>	The BLOCKID created in the BGPBLOCK table with a combination of prefix (if specified) and position of the midplane.
<machineSN>	The serial number defined in the BGPMACHINE table (usually `BGP`).
<psetsize>	The ratio of compute nodes to one I/O node (which depends on the hardware configuration).

The following command creates a block for each midplane on the system with a prefix of *Ex_* and the midplane appended to the end (see Figure 4-8). The *<psetsize>* suggests that this is an I/O rich rack (every node card has two I/O cards installed).

```
genblocks Ex_ BGP 16
```

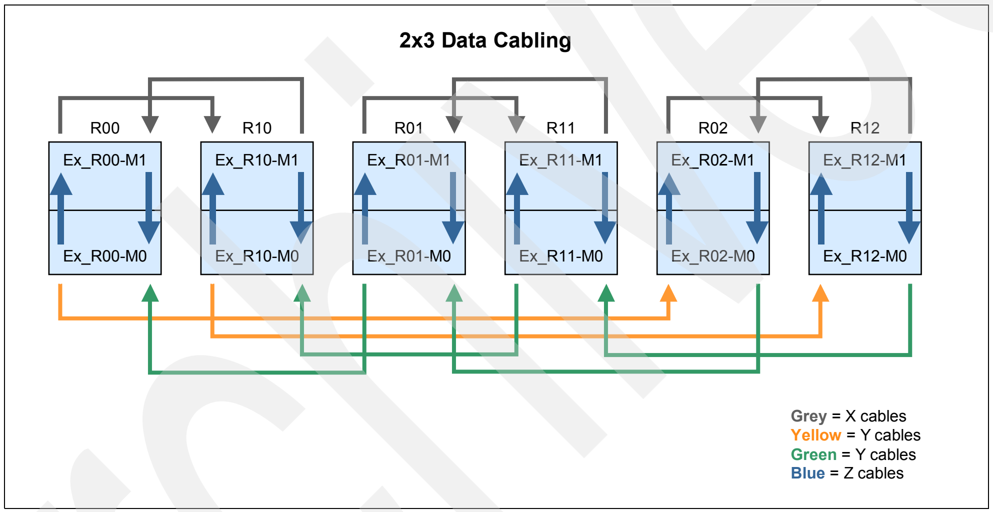


Figure 4-8 The genblock command creates a block from each midplane

gensmallblock

The **gensmallblock** command is used to generate a sub-base partition block. Small blocks can be created as 16, 32, 64, 128, or 256 node blocks. The **gensmallblock** command uses the following syntax:

```
gensmallblock <blockid> <midplane> <machineSN> <psetsize> <cnodes> <nodecard>
```

The parameter's definitions are as follows:

<blockid>	The name of the entry created in the BGPBLOCK table.
<midplane>	Specify the location of the midplane, for example `R00-M0`.
<machineSN>	The serial number defined in the BGPMACHINE table (usually `BGP`).
<psetsize>	The number of compute nodes to one I/O node (which depends on the hardware configuration).
<cnodes>	The number of compute nodes for the block (16, 32, 64, 128, or 256 can be used depending on the number of I/O nodes in the midplane).
<nodecard>	The starting node card of the compute nodes.

Note: When the <*cnodes*> parameter is equal to 16, the <*psetsize*> can be used to pass in the I/O node number (either 0 or 1).

The following commands create 32 node blocks on Rack 00, Midplane 0, node card J104, J111, J117, J203, and J214 called R00-M0_Jxxx_32. See Figure 4-9 for hardware that is used for each of the blocks. The <*psetsize*> suggests that each node card has two I/O nodes installed with every node attached to the functional network.

```
gensmallblock R00-M0_N01_32 R00-M0 BGP 16 32 N01
gensmallblock R00-M0_N04_32 R00-M0 BGP 16 32 N04
gensmallblock R00-M0_N07_32 R00-M0 BGP 16 32 N07
gensmallblock R00-M0_N08_32 R00-M0 BGP 16 32 N08
gensmallblock R00-M0_N14_32 R00-M0 BGP 16 32 N14
```

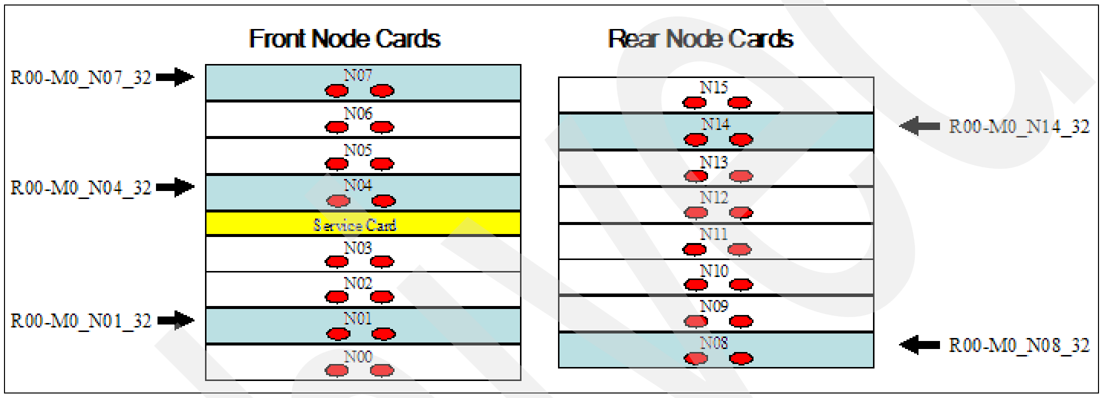


Figure 4-9 32 node small blocks

The following commands create four 128 node blocks on Rack 00 Midplane 0 (see Figure 4-10). The <*psetsize*> indicates that each node card has two I/O nodes installed and that all of the I/O nodes are attached to the functional network.

```
gensmallblock R00-M0_N00_128 R00-M0 BGP 16 128 N00
gensmallblock R00-M0_N04_128 R00-M0 BGP 16 128 N04
gensmallblock R00-M0_N08_128 R00-M0 BGP 16 128 N08
gensmallblock R00-M0_N12_128 R00-M0 BGP 16 128 N12
```

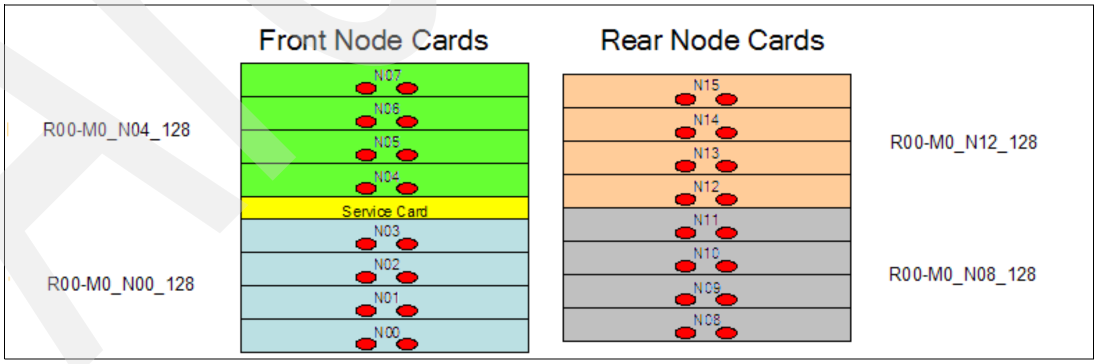


Figure 4-10 128 compute node block map

genBPblock

The **genBPblock** command is used to generate a torus block for a set of base partitions. The **genBPblock** command uses the following syntax:

```
genBPblock <blockid> <machineSN> <psetsize> <cornerBP> <xsize> <ysize> <zsize>
[xPT yPT zPT [splits]]
```

The parameter's definitions are as follows:

<blockid>	The name of the entry created in the BGPBLOCK table.
<machineSN>	The serial number defined in the BGPMACHINE table (usually `BGP`).
<psetsize>	The number of compute nodes to one I/O node (which depends on the hardware configuration).
<cornerBP>	The starting midplane that is used to create a multiple base partition block; use the POSINMACHINE value from the BGPMIDPLANE table (Rxxx).
<xsize>	The number of midplanes to configure in the X dimension.
<ysize>	The number of midplanes to configure in the Y dimension.
<zsize>	The number of midplanes to configure in the Z dimension.

xPT, yPT, zPT, and splits are optional arguments to specify the use of pass through and X-split cables.

The following command creates a block for only the Midplanes on Rack 00 called Ex_R00 (see Figure 4-11). The *<psetsize>* indicates that each node card has only one I/O card installed and has only one node attached to the functional network.

```
mmcs$  genBPblock Ex_R00 BGP 32 R00-M0 1 1 2
```

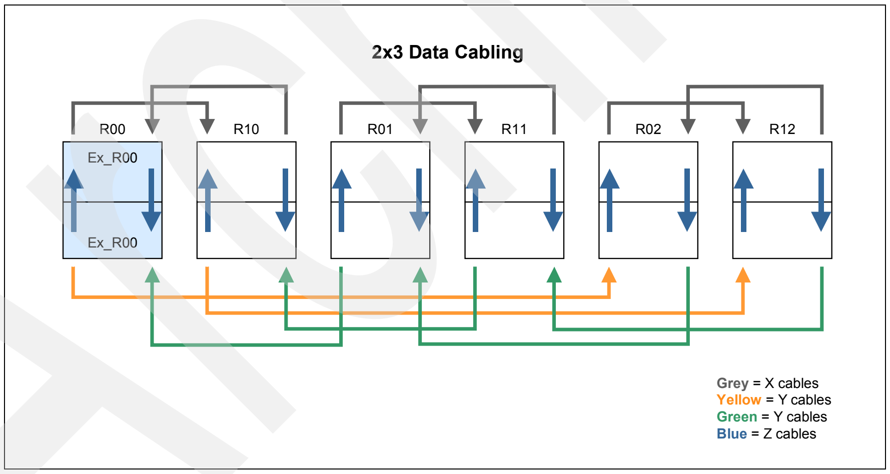


Figure 4-11 Creating a one rack block with genBPblock

Pass through

Recall from earlier discussions about data cables that the data only flows in one direction. If you look at the data cable flow in Figure 4-12, notice that the Y dimension is comprised of green and violet cables. For data to flow from R00 to R01, the data must pass through R02 and R03. Blue cables are used to connect the two midplanes in a rack (Z dimension).

In 4.2, "Displaying and creating blocks using Blue Gene Navigator" on page 103, we describe how you can create blocks using the Block Builder that uses the link cards in a midplane, without using the compute nodes in that midplane. The **genBPblock** command can also be used to create partitions that use either pass through, split cables, or both (for example, if we have a row with four racks in it, but we only want to use the first, second, and fourth racks in the row to create a block). In this example, we must use the link cards in the third rack to complete our torus. Figure 4-12 depicts this example block.

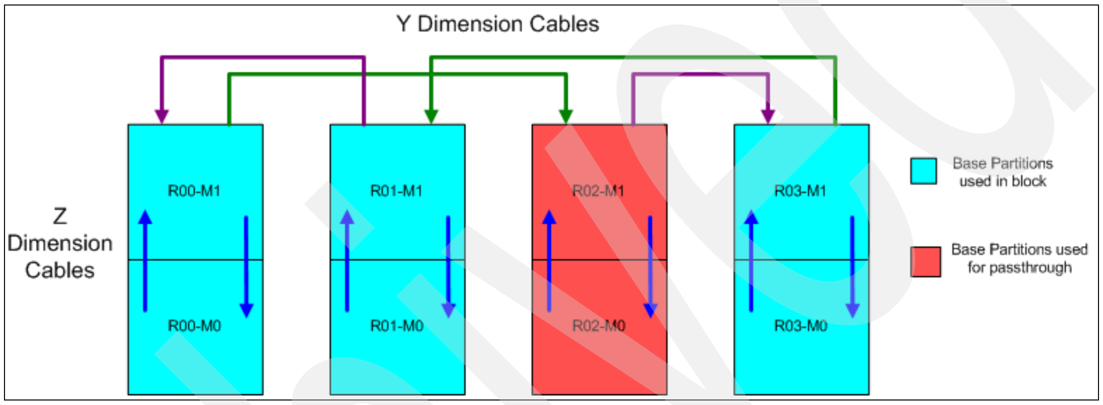


Figure 4-12 Three rack block using pass through

The command to generate this block is:

```
genBPblock 3Rack BGP 32 R00-M0 1 3 2 1 1011 11
```

This command indicates that we must start with R00-M0 and, going in the Y dimension, use the first, third, and fourth midplane in the cycle. The second midplane in the cycle (R02), because there is a zero in the second position of the Y specification (1011), is used only for pass through. Using the data cables, starting with R00 and moving in the Y+ direction, we start with R00, then go to R02, R03, and R01 before returning to the origin. So, in this example, the midplanes in rack R02 are used only for pass through. Then, a single rack block on rack R02 can be used simultaneously with block 3RACK, because they do not overlap.

Note: The specification of which midplanes to include and which are for pass through only, uses the order determined by the cabling, not the physical position of the racks.

Split data flow

Split data flow happens in the X dimension on large systems. Figure 4-13 shows the data flow for the X dimension. Data flows from R0 ∅ R2 ∅ R3 ∅ R1 ∅ R0. If we create a block that consists of Row0 and Row1 (not using the split data flow), we must pass through Row2 and Row3 in order for Row0 to access Row1.

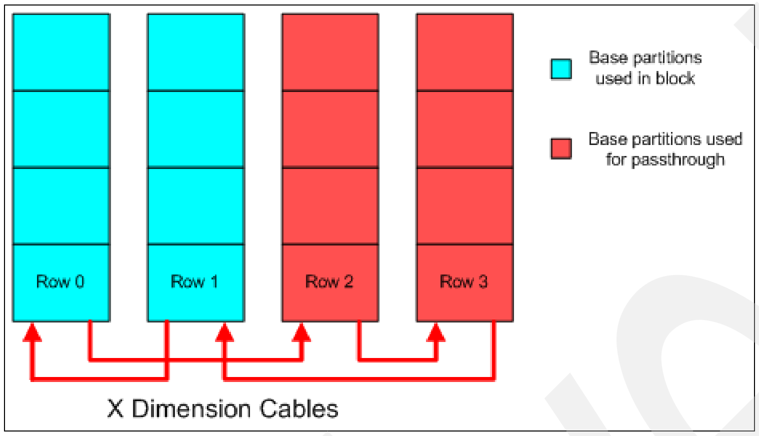

Base partitions used in block
Base partitions used for passthrough
Row 0
Row 1
Row 2
Row 3
X Dimension Cables

Figure 4-13 Data flow with X cables only

At the top of Figure 4-14 is the data flow when X-Split cabling is used. In this environment, if we create a block that consists of Row0 and Row1, we can use the X-Split cable to go to Row1, and return on the X cable. Rows 2 and 3 are not used in this partition.

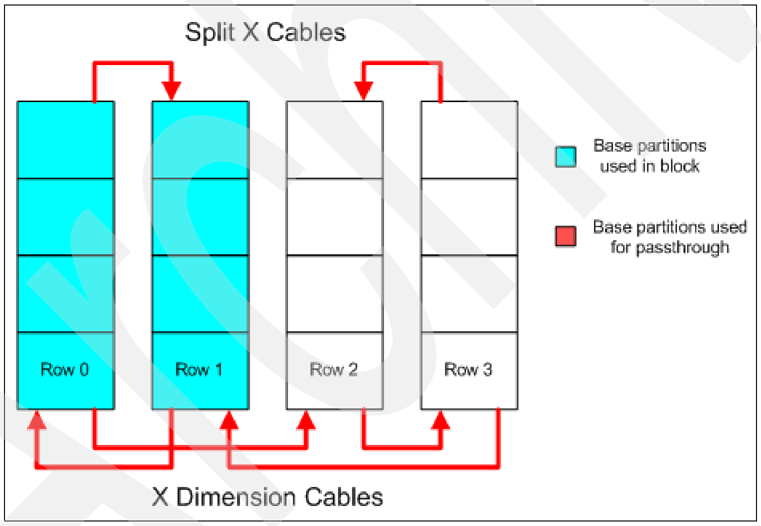

Split X Cables
Base partitions used in block
Base partitions used for passthrough
Row 0
Row 1
Row 2
Row 3
X Dimension Cables

Figure 4-14 Data flow with X and split X cables

You can determine if X-Split cables are used in the system by checking the Link Summary page in Navigator.

To create a block using Row 0 and Row 1 from the system shown in Figure 4-14 use the following command:

```
genBPblock R0R1 BGP 32 R00-M0 2 4 2 11 1111 11 SX
```

The addition of the last argument (SX) is the way to indicate the use of split cables. An S means that the cable coming out of row 0 is an x-split cable. The X in the next position means use a regular x cable coming out of row 1 to return to row 0.

genfullblock

The **genfullblock** command is used to generate a block for the entire system. The **genfullblock** command uses the following syntax:

```
genfullblock <blockid> <machineSN> <psetsize>
```

The parameter's definitions are as follows:

<blockid>	The name of the entry created in the BGPBLOCK table.
<machineSN>	The serial number defined in the BGPMACHINE table (usually `BGP`).
<psetsize>	The ratio of compute nodes to one I/O node (which depends on the hardware configuration).

The following command creates a block named *System* that uses every midplane on the system (see Figure 4-15):

```
mmcs$  genfullblock System BGP 8
```

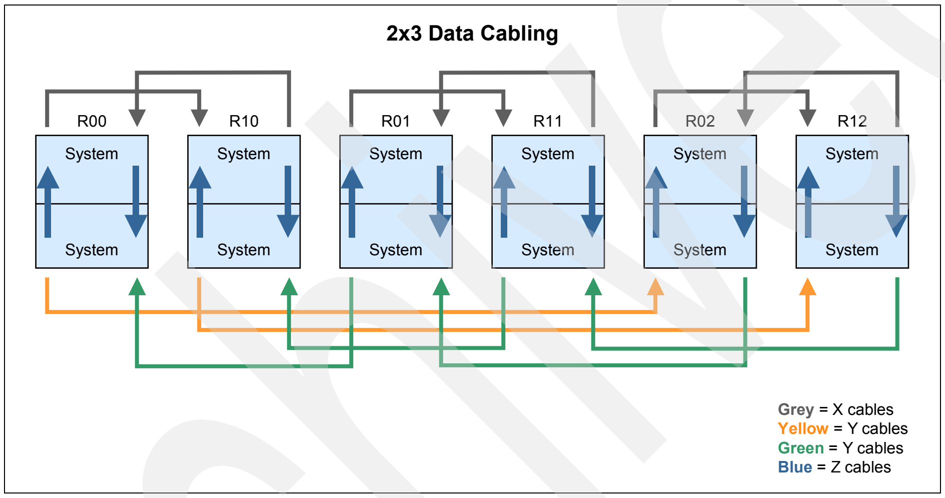


Figure 4-15 Block using all resources in the Blue Gene/P core

4.4 Deleting blocks

Blocks can be deleted from either the MMCS console or Navigator's MMCS interface. The syntax for the command is:

```
delete <bgpblock> <blockID>
```

The parameter's definitions are as follows:

<bgpblock>	The DB2 table that contains the block information for the system.
<blockID>	The unique identifier (name) of the block deleted.

5

Midplane Management Control System

You use the Midplane Management Control System (MMCS) to configure and allocate blocks of compute nodes and I/O nodes and to run programs on the Blue Gene/P system. Referred to as the *high-level control system*, MMCS runs on the service node and provides an interface to the hardware level system components. You can access the MMCS directly using the MMCS console or Blue Gene Navigator or through the various scheduler programs (using the scheduler APIs).

In addition to the high-level control functions of the MMCS server, the following functions are incorporated into the server:

- A relay that passes Reliability, Availability, and Serviceability (RAS) messages from the mcServer to the database
- Environmental monitoring

Evolution Note: Although much of the base MMCS from Blue Gene/L was maintained, there are differences in the Blue Gene/P version. One notable change is that MMCS no longer contains any low level, hardware debug functionality.

This chapter explains MMCS and how to use it.

5.1 MMCS components

This section describes the components of MMCS and how it serves as an interface between users and the Blue Gene/P hardware. Most users access MMCS primarily through a job management subsystem, such as LoadLeveler. However, system administrators generally want to use the cluster's resources more directly through **`mmcs_db`** commands.

Figure 5-1 shows an overview of the conceptual framework on which the **`mmcs_db`** command supports and operates.

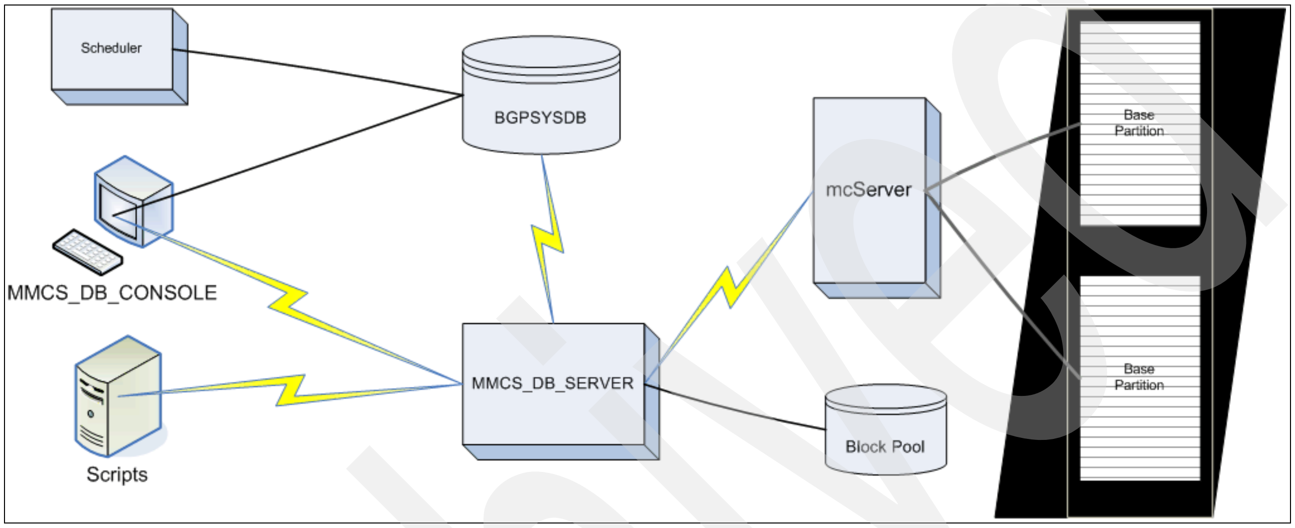


Figure 5-1   MMCS components

MMCS server handles requests from clients

The MMCS server runs on the service node and listens for incoming connections from MMCS clients or script clients. The MMCS server is capable of handling multiple client connections. It performs other functions simultaneously, such as monitoring for RAS events, environmental monitoring, and mailbox monitoring.

MMCS clients

MMCS commands are issued from either the MMCS console or from a program or script through a socket connection. The **`mmcs_db`** commands can be sent to the MMCS server from any program that can communicate over a socket, including most scripting languages. MMCS console supports additional commands for querying or updating the central database. The job management subsystem interacts with **`mmcs_db`** through a set of programming interfaces to the database.

Blocks are defined in the database

MMCS console and script clients allocate only predefined blocks configured in the central database. System administrators define these blocks based on the needs of the users. In that respect, LoadLeveler has a greater capability. It can analyze current cluster resources and define blocks dynamically based on specific job requirements and cluster resource availability.

The MMCS server performs the actual block allocation, booting, switch configuration, and deallocation based on requests from its clients. It keeps track of allocated blocks within the Blue Gene/P cluster. These blocks are not freed automatically when a client disconnects. They remain in a pool of allocated blocks and can be subsequently selected and operated on by other clients. Because the blocks are allocated to a user by name, any client that is self-identified as that same user is eligible to select and use an allocated block. MMCS console and LoadLeveler set the current user name automatically, but script clients must do this explicitly.

> **Note:** You can use the **`setusername`** or **`set_username`** MMCS commands to establish or change the identity of the current user.

mcServer communicates with the Blue Gene hardware

The clients are not aware of the existence of mcServer in the control system, but mcServer plays a key role in the process. Through mcServer, the MMCS Server can monitor the availability of the hardware. mcServer also controls access to the hardware to avoid conflicts. See 3.2, "mcServer" on page 85 for more details.

5.2 Starting the MMCS server

By default, the BGPMaster process starts the MMCS server. BGPMaster monitors the control system processes and attempts to restart them if they happen to end. See Chapter 3, "BGPMaster" on page 81 for more information.

You can start the MMCS server manually using the following command line:

```
/bgsys/local/etc/startmmcs
```

You can also use the **`/bgsys/driver/ppcfloor/bin/mmcs_db_server`** command.

The **`startmmcs`** command creates a log file using the following syntax:

```
<hostname>-mmcs_db_server-yyyy-mmdd-hh:mm:ss.log
```

It also creates `<hostname>-bgdb0-mmcs_db_server-current.log` as a symlink to the log file. The link is updated to link to the new file each time the server is started. If you use the **`mmcs_db_server`** command to start the MMCS server, no log file is generated.

> **Important:** Keep in mind that the MMCS server relies on the mcServer to maintain contact with the hardware. Thus, you must also start the mcServer manually if you are going to bypass BGPMaster. The **`startmcserver`** command (your local version) is also located in the `/bgsys/local/etc/` directory.

By default the MMCS server listens on port 32031 for incoming commands. The server listens only on the loopback interface (127.0.0.1), so only local clients can connect. The MMCS server communicates with the mcServer on port 1206.

5.3 Using MMCS console

This section details how to start and use the MMCS console. We cover some of the commonly used commands to demonstrate the use and output of the console.

5.3.1 Starting an MMCS console session

You use the `./mmcs_db_console` program to start a console session, and it is located in the `/bgsys/drivers/ppcfloor/bin` directory. By default no special authorities are required to run the `./mmcs_db_console` program.

Important: Accessing the MMCS server is only allowed from the service node. Anyone who has access to the service node can connect to the MMCS server using the MMCS console.

MMCS console initializes by connecting to the database and attempting to connect to the MMCS server. It does not continue if it cannot obtain a database connection but continues even if it cannot obtain an MMCS server connection, thus permitting access to the database while the server is not running.

5.3.2 Allocating blocks

Normally a user only uses a single console and a single block at a time. The user allocates the block using the **`allocate`** command, runs jobs using the **`submitjob`** command, and frees the block before ending the console using the **`quit`** command.

The **`allocate`** command establishes connections to the block after clearing any existing connections. It also reboots the block using the microloader, compute node, and I/O node images specified in the block definition in the database. Alternatively, you can use the **`allocate_block`** command which resets and establishes connections to the block but does not boot the block. You can then use the **`boot_block`** to boot a block allocated with **`allocate_block`**.

In some cases, a user might want to use the **`allocate`** command to allocate a block, use the **`submitjob`** command for a job, and then use **`quit`** while the job is running. To return and operate on the same block, the user starts MMCS server and uses **`select_block`** to reconnect to the previously allocated block.

There might be some situations in which a user wants two console sessions connected to the same block. For example, suppose a user started a **`waitjob`** command to wait for a program to end, and it does not appear to be ending. The console is hanging until the job ends, due to the **`waitjob`** command. The user could start a second console session, select the original block with **`select_block`**, examine the job status with **`list bgpjob`**, or stop the job by using the **`kill_job`** command.

5.3.3 Starting jobs

Running an application program with MMCS is done by using the database. The **`submitjob`** command adds a record to the BGPJOB database table and immediately returns. The MMCS server monitors the database. It starts jobs on initialized blocks by communicating with CIOD running on the block's I/O nodes. In this case, the CIOD is not aware of jobs that end and start within the block. The jobs are controlled by the master scheduler program.

When the job starts, the UID and GID are set to that of the user who submitted the job. Also the current directory is set to the one that is specified on the **submitjob** command. Any standard output is directed to text files in the directory specified on the **submitjob** command.

You can use the **wait_job** command to wait for the job to complete. You can also use the **list bgpjob** command to view the job's status, definition, and any error messages from job initialization.

5.4 MMCS commands

Table 5-1 contains a list of the supported MMCS commands for Blue Gene/P. The *HPC* column indicates whether the command is applicable to traditional High Performance Computing jobs. The *HTC* column indicates whether the command is applicable to High Throughput Computing jobs. For more details about HTC mode, see Chapter 6, "High Throughput Computing" on page 137.

Table 5-1 Supported commands

Command	HPC	HTC	Syntax and description
!			`! [shell command]` This command escapes to a subshell if no <command> is specified. Use "exit" to return to the mmcs$ prompt. If the optional <command> is specified, it runs that command in a subshell and returns you to the mmcs$ prompt. (Can only be used on the console)
#			`# [<comment>]` This command indicates a comment line.
<			`<` <*filename*> This command pipes command input from <*filename*> to MMCS server. (Can only be used on the console)
addbootoption	X	X	`addbootoption` <*blockId*|ALL> <*bootoption*> `addbootoption` is used to add a boot option for a specific block or all blocks. This value overrides entries in the db.properties file. Multiple options can be separated by commas, no spaces.
allocate	X	X	`allocate` <*blockId*> `[htc=<smp | dual | vnm | linux>]` The allocate command performs **select_block**, **allocate_block**, **boot_block**, and **wait_boot** for the specified block (block must be free). Using the command with the `htc=<smp | dual | vnm | linux>` argument boots the specified partition in HTC mode. If only a block name is specified, the default is to boot the block in HPC mode.
allocate_block	X	X	`allocate_block` <*blockId*> `[options]` `allocate_block` marks the block as allocated but does not boot it. **Options include:** ► `no_connect`: Do not connect to block hardware. ► `pgood`: Reset `pgood` on block hardware. ► `diags`: Enable block to be created when components are in Service state. ► `shared`: Enable nodecard resources to be shared between blocks. Implies no_connect. ► `svchost_options=<svc_host_configuration_file>` ► `htc=<smp | dual | vnm | linux>` : Allocate block in HTC mode
assignjobname	X		`assignjobname` <*jobId*> <*jobName*> Assigns a character job name to a specific job ID. (Can be used only on the console.)

Command	HPC	HTC	Syntax and description
`associatejob`	X		`associatejob <`*`jobId`*`> <`*`blockId`*`>` Associates a previously created job with a pre-allocated block. (Can be used only on the console.)
`boot_block`	X	X	`boot_block [update] [<`*`options`*`>]` Initialize, load and start the block's resources. **Options include:** ▶ `uloader = <path>`: The microloader. ▶ `ioload = <path>`: The I/O node elf images. A comma separated list. ▶ `cnload = <path>`: The compute node elf images. A comma separated list. ▶ `virtual_node_mode`: Prepare the nodes to run the application on all cores. ▶ `no_global_interrupts`: Do not use global interrupts ▶ `standalone`: Disable CIOD interface ▶ `boot options`: Any other options are passed to the machine controller on the **`boot`** command **Scenarios:** ▶ If no options are specified, the `uloader`, `ioload`, `cnload`, and `boot options` specified in the block definition are used. ▶ If any options are specified, and `update` is not specified, only the `uloader`, `ioload`, `cnload`, and `boot options` specified on the **`boot_block`** command are used. ▶ If `update` is specified, the `uloader`, `ioload`, `cnload`, and `boot options` from the **`boot_block`** command are combined with the block definition. Any boot options are added to those from the block definition. Any `uloader`, `ioload`, or `cnload` specified on the boot_block command replaces the specification from the block definition, but any `uloader`, `ioload`, or `cnload` not specified on the **`boot_block`** command will be taken from the block specification.
`connect`	X	X	`[<`*`target`*`>] connect [`*`options`*`]` Connect to a set of a block's resources. **Options include:** ▶ `targetset=[ temp | perm ]`: Create the mcServer target set as temporary (default) or permanent. ▶ `outfile=<`*`filename`*`>`: Direct the console messages to a file. ▶ `rasfile=<`*`filename`*`>`: Direct the RAS messages to a file. ▶ `tee`: Direct the mailbox out to both the file and stdout. ▶ `no_ras_disconnect`: Do not disconnect the block on a fatal RAS event. ▶ `pgood`: Reset pgood on block hardware.
`copyblock`	X	X	`copyblock <`*`existingblockId`*`> <`*`newblockId`*`>` Duplicate an existing block, along with all of its boot information.
`createjob`	X		`createjob [blockId] <`*`executable`*`> <`*`outputDir`*`> [mode]` Defines a job in the BGPJOB table that runs <*`executable`*> in optional initialized <*`blockId`*>. `createjob` is used in conjunction with **`setjobargs`** and **`startjob`**. The `<executable>` parameter specifies the path to the program to be started. <*`outputDir`*> specifies a writable directory to become the home directory for the job and to contain the output files for text written to stdout and stderr. The job is in queued status and is not eligible to run until **`startjob`** is issued. (Can be used only on the console.)
`debugjob`	X		`debugjob <`*`jobId`*`>` Starts the debugger on a running job.
`delete`	X	X	`delete <`*`db2table`*`> <`*`id`*`>` Deletes the specified record ID from the database. The only two valid tables for this command are BGPBLOCK and BGPJOB. (Can be used only on the console.)

Command	HPC	HTC	Syntax and description
deselect_block	X	X	`deselect_block` Stop working with the currently selected block.
disconnect	X	X	`disconnect` Disconnect from a set of block resources.
free	X	X	`free <blockId>` Releases the resources that are associated with the *blockId*. Marks the block free in the BGPBLOCK table. Before an HTC partition can be freed, no jobs can be active on the partition. To end all running HTC jobs on a partition use the **kill_htc_jobs** command.
free_all	X	X	`free_all` Releases all blocks owned by the console user.
free_block	X	X	`free_block` Releases the resources that are associated with the currently selected block. Marks the block free in the BGPBLOCK table. Before an HTC partition can be freed, no jobs can be active on the partition. To end all running HTC jobs on a partition, use the **kill_htc_jobs** command.
genBPblock	X	X	genBPblock *<blockId> <machineSN> <psetsize> <cornerBP> <xsize> <ysize> <zsize>* [xPT yPT zPT [splits]] Generates a block for a set of base partitions (BP). The *<psetsize>* is the number of compute nodes for each I/O node (see Chapter 4, “Blocks” on page 101 for more information about creating blocks). For *<cornerBP>* specify the midplane by location, that is, Rxx-Mx of the first BP in the block. The size is provided in terms of number of midplanes in X, Y, and Z dimensions. The *<cornerBP>* is in the 0,0,0 position of the midplanes that make up the block. Except for the *<cornerBP>*, the midplanes included in the generated block depend on the X, Y, Z cabling of your machine. xPT, yPT, zPT, and splits are optional arguments to specify the use of pass through and X-split cables.
genblock	X	X	genblock *<blockId> <midplane> <machineSN> <psetsize>* Creates a block for the specified midplane.
genblocks	X	X	genblocks [blockIdprefix] *<machineSN> <psetsize>* Generates a block for each midplane in the system. If the blockprefix parameter is not supplied, the block name is the same as the position of the midplane (Rxx-Mx).
genfullblock	X	X	genfullblock *<blockId> <machineSN> <psetsize>* Generates a single block for the entire machine.
gensmallblock	X	X	gensmallblock *<blockId> <midplane> <machineSN> <psetsize> <cnodes> <nodecard>* Creates a sub-midplane block. The valid number of compute nodes (*<cnodes>*) is 16, 32, 64, 128, or 256.
getblockinfo	X	X	getblockinfo *<blockId>* Returns the boot information, boot mode and status of the specified block.
getjobinfo	X	X	getjobinfo *<jobId>*
getjobs	X	X	getjobs <jobs.xml> *<blockId>* Retrieves a list of jobs that are associated with the *blockId* identified in the command. The results are stored in the XML file.
help	X	X	`help [command name]` Prints the command summaries for all commands or, if named, a specific command.
job_status	X	X	job_status *<jobid>* [full] Display the status of the compute nodes running a job.

Command	HPC	HTC	Syntax and description
job_trace	X	X	`job_trace [blockId] off control data ciod-control ciod-data ciod-debugger ciod-fileio ciod-treeio ciod-test ciod-info full` Enables tracing of communications between mmcs and ciod. Job trace might be enabled globally or for a specific block. If no block is specified, it is enabled globally and all new jobs have tracing enabled regardless of how they are set by previous invocations of job_trace. [blockId] identifies a block in the BGPBLOCK table. Job tracing is not enabled on current jobs. The next new job for a trace enabled block is traced. Trace output is dumped to the mmcs log. Options are `control` for control data tracing only, `data` for stdout/stderr data tracing, and `off` to disable. Other options are `ciod-data`, `ciod-control`, `ciod-debugger`, `ciod-fileio`, `ciod-treeio`, `ciod-test`, and `ciod-info`. These options control trace elements for ciod. See ciod documentation for explanation.
kill_htc_jobs		X	`kill_htc_jobs [block=<`*`blockId`*`] [user=<`*`username`*`>] [timeout=<`*`seconds`*`>]` Kills all HTC jobs that are running by the supplied user and block. You must specify block or user name if no block is selected. If a block is currently selected, all jobs on it are killed matching the optional user name argument.
kill_job	X	X	`kill_job <`*`jobId`*`> [timeout]` Kills the specified job on the currently selected block. The optional `timeout` parameter overrides default specified by mmcs server. The command returns before the job is killed. You can use **`wait_job`** to wait for the job to be terminated.
killjob	X	X	`killjob <`*`blockId`*`> <`*`jobId`*`>` Kills the job on the specified blockId. The command returns before the job is killed.
list	X	X	`list <`*`db2table`*`> [<`*`value`*`>]` Returns the contents of the database table listed. The following are the only valid table/field combinations: ► bgpnode / location ► bgpnodecard / location ► bgpmidplane / location ► bgpmachine / serialnumber ► bgpblock / blockid (wild card with %) ► bgpproducttype / productid ► bgpjob / jobid ► bgpjob_history / jobid ► bgpeventlog / block (Can be used only on the console.)
list_blocks	X	X	`list_blocks` Returns all of the currently allocated blocks. The output includes the user, number of consoles started and if the output is redirected to the console.
list_bps	X	X	`list_bps` Returns base partition information for the entire machine, including blocks that are booted on each of the midplanes.
list_htc_jobs		X	`list_htc_jobs [block=`*`blockid`*`] [pool=`*`poolid`*`] [user=`*`username`*`]` Prints the job ID, status, user name, block ID, pool, location, and exe for all active high throughput computing jobs. Omitting a block, pool, and user name shows all active jobs. Adding either one narrows the query.
list_jobs	X	X	`list_jobs [<`*`username`*`>]` Prints the job ID, status, user name, block ID, and executable for all jobs that are in the BGPJOB table or those jobs in BGPJOB with optional <*`username`*>. Jobs that are running in HTC mode are also noted.

Command	HPC	HTC	Syntax and description
`list_users`	X	X	`list_users` Lists all mmcs users. The output returned includes thread number, block ID and if the output is redirected to the console.
`locate`	X	X	`[<`*`target`*`>] locate [neighbors] [ras_format]` Lists physical location (midplane, board, card, slot) of all nodes in allocated block. If target node is specified, lists the location of the specified node. Add [neighbors] to get the + and - neighbors in x,y,z dimensions for the specified node. Add [`ras_format`] to get location information in the format found in RAS events.
`locaterank`	X		`locaterank <`*`jobId`*`> <`*`mpirank`*`>` Locate a node using the jobId of a job and the MPI rank. Finds the job <*`jobId`*> in the job table or job history table, and using the attributes of the job, such as the block and the MPI mapping options, displays the physical location of the node with MPI rank <*`mpirank`*>.
`quit`	X	X	`quit` Exit from the MMCS console session. (Can only be used on the console)
`reboot_nodes`	X		`[<`*`target`*`>] reboot_nodes [<`*`blockid`*`>] [<`*`timeout`*`>]` Reboot compute nodes or specific I/O nodes. The *`blockid`* and *`timeout`* parameters are optional. If used, *`blockid`* must precede *`timeout`*. The **`reboot_nodes`** command displays an error message when issued against a block booted in HTC mode.
`redirect`	X	X	`redirect <`*`blockId`*`> on | off` Redirect I/O node output for the specified block to the mmcs console. Directs subsequent mailbox output back to the socket connection that this command is received on. Allocating or freeing the block will stop the mailbox redirection.
`replyformat`	X	X	`replyformat [0 | 1]` With no parameters replyformat displays the current reply format, 0 or 1. ► *0* means that each reply is returned as a single text line, embedded new lines are indicated by a semicolon and then end of the reply is indicated by a new line. ► *1* means that each reply is returned as one or more text lines, each line terminated by a new line. The end of the reply is indicated by a single quotation mark (').
`select_block`	X	X	`select_block <`*`blockId`*`>` Select an already allocated block with which to work. `select_block` does not initialize the block nor reset connections. This command is intended to be used to reconnect to an allocated block from a another mmcs_db console session.
`set_username`	X	X	`set_username <`*`username`*`>` Set user name for block allocation and job scheduling. You cannot issue commands against a block that is allocated to another user. When the MMCS console starts, this command is issued automatically on behalf of the current user and so does not need to be issued by the MMCS console user unless they want to run under a different user ID. However, it must be issued as the first command by any mmcs_db script.
`setblockinfo`	X	X	`setblockinfo <`*`blockId`*`> <`*`mloader`*`> <`*`cnload`*`> <`*`ioload`*`> [<`*`bootoptions`*`>]` Sets the boot images for a block. To set multiple loads for *`cnload`* or *`ioload`*, use multiple paths, separated by commas, with no spaces.
`setbootoptions`	X	X	`setbootoptions <`*`blockId`*` | ALL> <`*`bootoptions`*` | NULL>` Sets boot options for a block (*`blockId`*) or globally (ALL). To set multiple options separate by commas (no spaces).

Command	HPC	HTC	Syntax and description
setdebuginfo	X		`setdebuginfo <jobId> <debugger> [<args>]` Set the debugger image and optional debugger arguments for a specific job.
setjobargs	X		`setjobargs <jobId> <args...> <envs....>` Set arguments and environment variables for a specific job. ► *args* is a list of arguments passed to the program. ► *envs* is a list of environment variable settings of the form `name=value`. The environment variables are accessible to the program when it is running on a compute node. (Can be used only on the console)
setusername	X	X	`setusername <username>` Set user name for block allocation and job scheduling. You cannot issue commands against a block that is allocated to another user. When the MMCS console starts, this command is automatically issued on behalf of the current user, and so does not need to be issued by the MMCS console user unless they want to run under a different user ID. However, it must be issued as the first command by any mmcs_db script.
signal_job	X	X	`signal_job <jobid> <signal>` Schedule the specified job on the selected block to be signaled with the signal contained in the command. The command returns before the job is signaled.
signaljob	X	X	`signaljob <blockid> <jobid> <signal>` Schedule the specified job on the block indicated to be signaled with the specified signal. The command returns before the job is signaled.
sleep	X	X	`sleep <seconds>` Pause for the number of seconds specified in the command.
sql	X	X	`sql <sql-string>` OR `execute <sql-string>` This command can be used to run an SQL statement from the MMCS console. It cannot be used to run a statement that returns a value (that is, queries)
starthwpolling	X	X	`starthwpolling [ncseconds [scseconds [lcseconds]]]` Start polling the hardware for environmental readings. See 5.7, “Environmental Monitor” on page 129
startjob	X		`startjob <jobId>` Schedules the specified job `<jobId>` to be started. Marks the job as eligible to start in the BGPJOB table. The job will not be started until the block is available and no prior jobs for the block are eligible to start. (Can be used only on the console)
status			`[<target>] status` Lists nodes in allocated block, the type of each node, and whether it is running a program or vacant. If the block has been booted, it will show that a program is running.
stophwpolling	X	X	`stophwpolling` Stop polling the hardware for environmental readings. See 5.7, “Environmental Monitor” on page 129

Command	HPC	HTC	Syntax and description
submit_job	X		submit_job <*executable*> <*outdir*> [mode] <*args*> <*envs*> Starts an <*executable*> running on the currently selected block. ► <*executable*> specifies the path to the program to be started. ► <*outdir*> specifies a directory to become the home directory for the job and to contain the output files for text written to stdout and stderr. The bgpadmin user *must* have permission to access this directory either because it is a world executable or because the bgpadmin group owns it and it is group executable. ► [mode] is optional. The possible modes are SMP, DUAL or VN. If omitted, the job will run in SMP mode. ► <*args*> is a list of arguments passed to the program. ► <*envs*> is a list of environment variable settings of the form name=value. The environment variables will be accessible to the program when it is running on a compute node. The response to this command contains a job number which can be used with waitjob or killjob. **submit_job** is not allowed for HTC jobs.
submitjob	X		submitjob <*blockId*> <*executable*> <*outdir*> [mode] [args] [envs] Create a job in the DB to run <*executable*> in <*blockId*> and schedule it to start. This is the same command as submit_job, but with an additional BlockId parameter. This form of the command causes the specified block to be selected before command is executed. See **submit_job** for full description of parameters. **submitjob** is not allowed for HTC jobs.
username	X	X	username Returns the current user name.
version	X	X	version Displays the current MMCS version.
wait_boot	X	X	wait_boot [<*minutes*>] Wait for the selected block to complete booting before returning control. <*minutes*> specifies the maximum time to wait, in minutes. The default is 15 minutes. The command does not complete until the block is fully initialized to the point that mmcs can communicate with CIOD on all of the I/O nodes or the wait time has expired
wait_job	X	X	wait_job <*jobId*> Waits for the job with the specified jobId to end on the selected block. Returns immediately if the job is no longer running or failed to start.
waitjob	X	X	waitjob <*blockId*> <*jobId*> Waits for the job with the specified jobId to end on the specified block. Returns immediately if the job is no longer running or failed to start.
write_con	X	X	[<*target*>] write_con <*console-command*> Send <*console-command*> to target node for execution. Output will be returned to mailbox (either console or I/O node log).

Table 5-2 contains commands that accept an optional <*target*>, which can be used to specify the nodes on which to apply the operation.

Table 5-2 Node commands

Target expression	Meaning
{*}	all nodes (default)
{i}	all I/O nodes
{c}	all compute nodes

Target expression	Meaning
{nc}	all node cards
{lc}	all link cards
{l}	all link chips
{<N>}	single node
{<N>,<N>...}	list of nodes
{<N>-<N>}	subrange of nodes
{<N>-<N>,<N>,...}	combination of range and list

Individual cores can be specified by appending a period followed by the core number to the target. For example core 0 of node 1 would be as follows:

```
{1}.0
```

5.5 Output

The block control functions run as threads under the MMCS server process on the service node. Therefore, any output that is directed to standard output or standard error is directed to the MMCS server process console.

5.5.1 Command and response

The response to an **mmcs_db** command consists of the word `OK` followed by optional messages or the word `FAIL` followed by an error message (see Figure 5-2). The internal representation of the response string depends on the reply format in effect at the time.

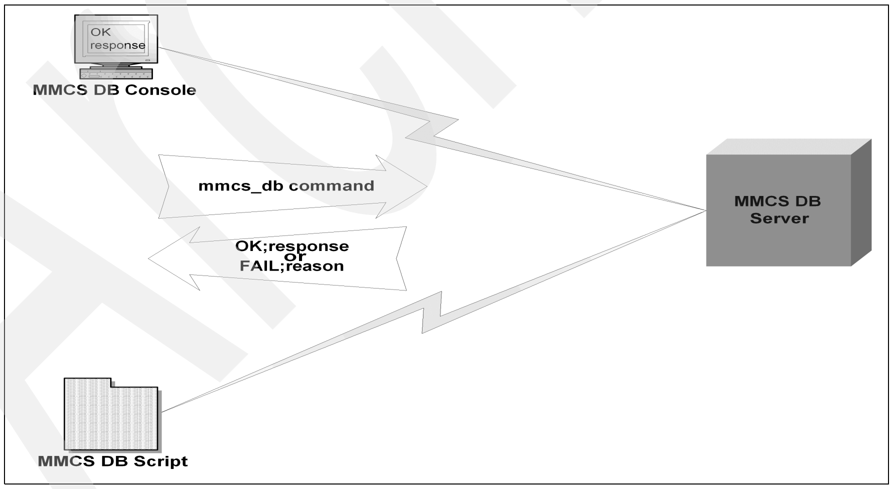


Figure 5-2 Command and response (format 0)

The replyformat command

The **`replyformat`** command is used to set the response received from the MMCS server. The command takes a single, optional argument:

`replyformat 0`	Establishes the single-line reply mode (the old format).
`replyformat 1`	Establishes the multi-line reply mode (the new format).
`replyformat`	(No parameters) Displays the current reply mode.

The old format has a 16 KB length limit on all replies. If the limit is exceeded, the reply is truncated with the message:

```
   ;ABORT;reply length exceeded
```

MMCS server defaults to the newer multi-line format. However, scripts default to the old single-line format to maintain compatibility.

replyformat 0

This response consists of a single string, terminated by a single new line character at the end of the string. The first word in the string is either `OK`, indicating success, or `FAIL`, indicating failure. If additional data follows, `OK` or `FAIL` is followed by a semicolon (;) and the additional data. For example, the response to a successful submitjob command is as follows:

```
   OK;jobId=xx\n
```

Multiline responses have embedded new line characters that are replaced with semicolons. Each semicolon represents a new line character that was replaced prior to being sent over the socket connection from **`mmcs_db`** to the client. The MMCS server replaces these semicolons with new line characters before writing them to the console window, so a multiline response looks the way it does if it is printed from the local process. However, it is the script's responsibility to parse the response and (optionally) replace the semicolons with new line characters.

replyformat 1

The response content is the same as with reply format 0. Only the format differs. The response consists of `OK` or `FAIL` in the first line, optionally followed by additional data. The response is sent as multiple newline-terminated strings. There is no limit to the number of strings that can be sent in the reply.

The last line is followed by a line consisting of the null character followed by a newline (`\0\n`).

5.5.2 Mailbox output

Each node on an allocated block has a mailbox associated with it, and the block's thread in MMCS server monitors the mailbox for output. Output from programs running on the compute nodes and I/O nodes can be written to the mailbox and processed by the mailbox thread.

Mailbox output is the I/O node output, which is primarily debug and command output from I/O node kernel and compute node kernel, because application program output is handled separately.

The MMCS server has various options for directing this mailbox output:

- Default

  If you do nothing, the mailbox output is directed to the MMCS console. If you have redirected the MMCS server standard output to a file, you can view the mailbox output in that file. However, the MMCS server output contains data from potentially many mailboxes.

- Outfile parameter

  You can specify `outfile=filename` on the **`allocate`** and **`allocate_block`** commands to have the mailbox output directed to a file.

- Redirection

  The MMCS server has a **`redirect`** command that sends the mailbox output back over the socket connection.

5.5.3 Job output

In MMCS the **`submitjob`** command is used to start an application program. It causes the program's standard output to be directed to a file. This file is created in the directory specified on the **`submitjob`** command. It is named with the block name, job number, and output stream, such as R00-M0-26.stdout.

5.5.4 Redirection

The **`redirect`** command is an exception to the response format rule given previously. The **`redirect`** command causes the MMCS server mailbox thread to send all of its output back over the same socket connection. The mailbox data is sent over the connection as it is received from the mailbox. The client issuing the **`redirect`** command loops immediately and reads the **`mmcs_db`** socket connection. Semicolons should not be replaced with new line characters. The socket connection is reset when the mailbox thread is terminated by an **`allocate`**, **`allocate_block`**, or **`free`** command.

Note: The **`redirect`** command is not available in the Navigator's version of the MMCS console. After a block is allocated, or **`select_block`** is used, a button displays that allows you to view the selected block's output.

5.6 CIOD

Evolution Note: In Blue Gene/P, the MMCS server monitors both blocks and jobs. In Blue Gene/L the MMCS server blocks functions, and a separate process called CIODB monitors the database for job requests and communicates with the I/O nodes.

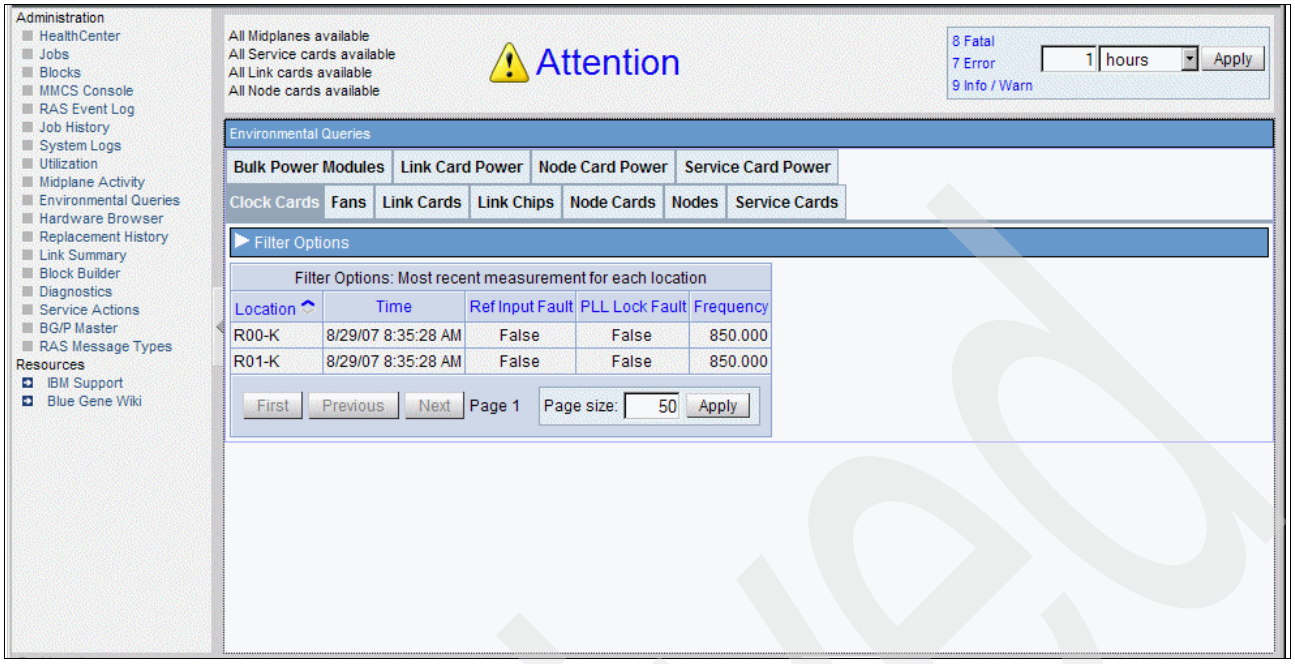
CIOD is a user-mode daemon that runs on the I/O node and provides a bridge between the compute nodes and the outside world. CIOD takes messages from multiple input sources, processes the message, and returns a result. CIOD accepts messages from three sources:

- The control system sends messages using the CioStream and DataStream protocols on two socket connections from the service node. The control system sends messages to load and start a new job, send standard input, and send a signal to a running job. CIOD sends standard output, standard error, and messages to report status.
- Compute nodes send messages using the CIO protocol on virtual channel 0 of the tree device. The messages are for both control, function shipping I/O, and debugging.
- An external debug server sends messages using the CioDebug protocol on pipes from another I/O node process.

For more information about CIOD on the I/O nodes, see Chapter 12, “Configuring I/O nodes” on page 213.

5.7 Environmental Monitor

The Environmental Monitor is another facet of MMCS. Monitoring on Blue Gene/P consists of collecting data from the running hardware, such as temperatures, clock frequency, fan speeds, and voltages. Generally referred to as *environmentals*, this information can be useful in indicating system health or in debugging problems. Environmentals are collected automatically every 5 minutes by default. The values collected are stored in the database for 3 months. Current and historical data can be viewed in Blue Gene Navigator by clicking **Environmental Queries** as shown in Figure 5-3. To assist in the efficient administration of the system, the Environment Monitor also generates RAS events when abnormal readings are encountered.



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

Filter Options: Most recent measurement for each location

Location	Time	Ref Input Fault	PLL Lock Fault	Frequency
R00-K	8/29/07 8:35:28 AM	False	False	850.000
R01-K	8/29/07 8:35:28 AM	False	False	850.000

Figure 5-3 Environmental Queries

5.7.1 How the Environmental Monitor works

The Environmental Monitor starts when the MMCS server is started. The MMCS server starts the envMonitor thread. In turn, the envMonitor thread starts individual threads to monitor each of the card types and a separate thread to purge stale data from the database. The threads that monitor hardware read the environmental data from mcServer and the Machine Controller (MC) and write the results to the database. By default, each of these threads sleeps for 5 minutes. The thread responsible for purging data is configured to sleep for 30 seconds, at which time it wakes up and checks for a kill request from the MMCS server. The types of cards monitored are Service cards, Link cards and Node cards. The data for the Bulk Power Modules, Fan Modules, and Clock cards are collected from the Service card.

Note: The lower midplane (midplane 0) is considered to be the master. The data for the Bulk Power Modules and the Clock Card is collected by the service card in the bottom midplane.

Multiple threads allow flexibility with the polling intervals for different types of hardware. The default values can be altered by adding (or changing) entries in the db.properties file. There are situations that might arise when you would want to disable the monitoring functions temporarily. As a general rule, before changing or disabling the monitor, check with IBM support.

Attention: You can use a range of between 60 and 1800 seconds to override the default polling interval.

One instance that you would need to stop polling briefly would be if you need to reset the polling intervals for a specific type of hardware. The monitoring can be stopped by issuing the `stophwpolling` command from the MMCS console without impacting users on the system.

The alternative is to make permanent changes by updating the db.properties file, and then restarting mmcs_db_server. The two possible methods to alter the polling intervals are as follows:

- For a permanent change update one or more of the following intervals in the db.properties file:

Service Cards	`mmcs_envs_sc_interval_seconds=`*`xxx`*
Link Cards	`mmcs_envs_lc_interval_seconds=`*`xxx`*
Node Cards	`mmcs_envs_nc_interval_seconds=`*`xxx`*

  As mentioned earlier, the default interval is hard coded into the monitor code with a value of 300 seconds. The first time that you change the values, you need to add the variables to the db.properties file. After you update the values restart the monitor by restarting the mmcs_db_server using the following command (as bgpadmin):

  ```
  bgpmaster restart mmcs_db_server
  ```

  Using this method overrides the default settings each time that the monitor is started.

- You can make a temporary change to the values for the current session by stopping and then starting the monitor with the new intervals specified in the start command.

  As long as the monitor is not stopped, the values that you supply by this method are used. First, you stop the monitor using the **`stophwpolling`** command at an MMCS prompt. The **`stophwpolling`** command may take a long time to complete since it requires that all polling threads complete their next poll. Then, when you restart the polling with the **`starthwpolling`** command, you supply up to three numeric values to control the rate at which the monitor polls. The parameters are `ncseconds`, `scseconds`, and `lcseconds`, which represent the number of seconds to wait in between each polling of the node card, service card, and link card, respectively.

  The following example of the **`starthwpolling`** command updates the node card data every 60 seconds, the service card every 600 seconds, and the link cards once every 120 seconds:

  ```
  mmcs$ starthwpolling 60 600 120
  ```

  The parameters are dependent on their position in the command, so you need to provide a value in each of the positions. The new values will take effect immediately and will be reset to either the default value, or the intervals set in the db.properties file on the next restart of the mmcs_db_server.

5.7.2 Blue Gene/P environmental database tables

Under normal circumstances the Navigator is the means of viewing environmental data on the Blue Gene/P hardware. Because the data is stored in the hardware database, you can also access it directly by running queries on the following tables:

- BGPBULKPOWERENVIRONMENT
- BGPCLOCKCARDENVIRONMENT
- BGPFANENVIRONMENT
- BGPLINKCARDENVIRONMENT
- BGPLINKCARDPOWERENVIRONMENT
- BGPLINKCHIPENVIRONMENT
- BGPNODECARDENVIRONMENT
- BGPNODECARDPOWERENVIRONMENT
- BGPNODEENVIRONMENT
- BGPSERVICECARDENVIRONMENT
- BGPSRVCCARDPOWERENVIRONMENT

The tables that are associated with the Environmental Monitor use both location and time as the primary key. See Appendix B, “Blue Gene/P hardware naming convention” on page 239 for a complete description of locations.

RAS events that are generated by the Environmental Monitor are written to the BGPEVENTLOG table.

Note: Each of the rows in the BGPEVENTLOG have a timestamp. There might be a delay between when an error occurs and when the Environmental Monitor polls the hardware. The time that is entered into the database is the actual time of the event, not when the hardware was polled.

Table 5-3 lists the events that generate an RAS event in the BGPEVENTLOG table.

Table 5-3 RAS event triggers

Hardware	**Event**
Service Card	Card cannot be contacted
Bulk Power Modules	► Module not contacted ► Module not responding ► General warning ► General fault ► Temperature warning ► Temperature fault ► Current warning ► Current fault ► Voltage fault ► Output 0 not enabled ► Output 0 fault
Fan Modules	► Fan Module not contacted ► Fan not responding ► Temperature limit exceeded ► Fan failed ► Fan speed out of spec (Actual speed 10% above or below desired speed)
Node Cards	► Card not contacted ► Over temperature
Link Cards	► Card not contacted ► Over temperature

5.7.3 Resources required

Based on the number of tables and the type of data that is collected for the various pieces of hardware, you can make a few general assumptions about the amount of storage that is required to support the Environmental Monitor. Table 5-4 shows an estimate of the amount of storage that is required using the default 5 minute polling period.

Table 5-4 Approximate storage requirements

Hardware	1 Day (per rack)	7 Days (per rack)	30 Days (per rack)	90 Days (per rack)
Link Cards and Link Chips	~700 Kb	~4.9 Mb	~21 Mb	~63 Mb
Node Cards and Nodes	~10 Mb	~70 Mb	~300 Mb	~900 Mb
Service Cards	520 Kb	~3.64 Mb	~15.6 Mb	~46.8 Mb
Total	~11.2 MB	~78.54 Mb	~336.6 Mb	~1 Gb

By default, the data that the Environmental Monitor collects is purged every 3 months. You can change this value by updating the `mmcs_envs_purge_months` entry in the db.properties file.

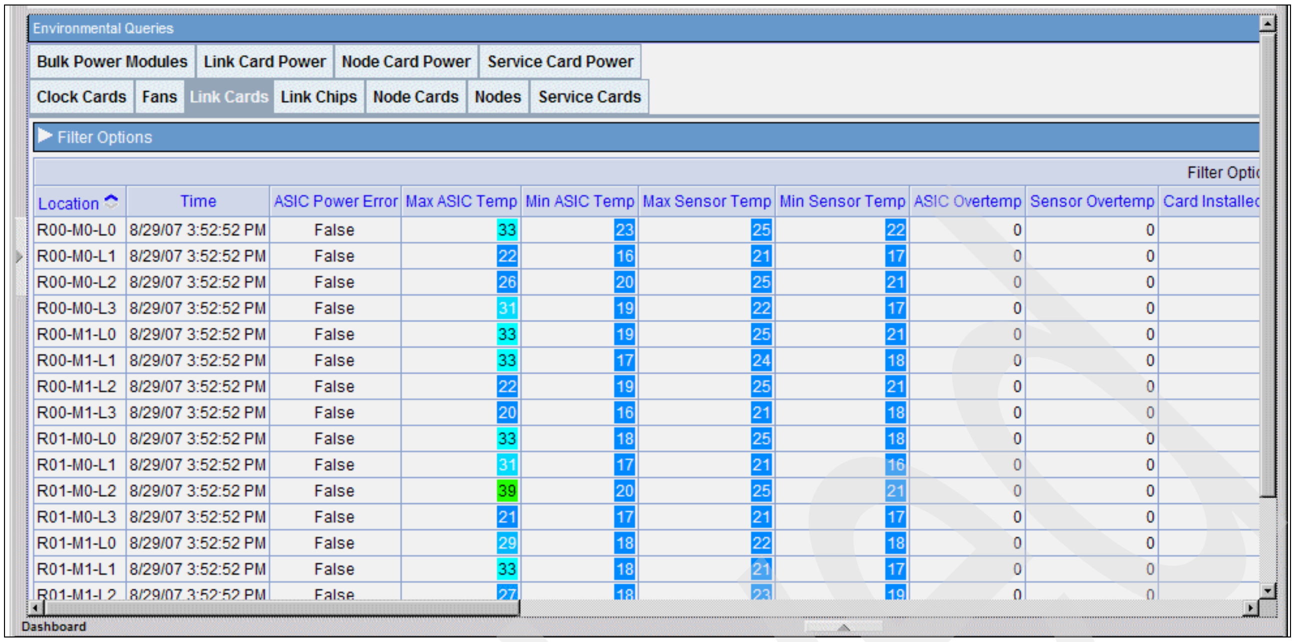
5.7.4 Blue Gene Navigator

You can view the environmental results, both current and historical, through the Navigator interface. Figure 5-3 on page 130 shows that there are tabs for all of the hardware that we discuss in this section. Each of the tabs reports a variety of data. For example, Figure 5-4 shows the results for the Link Cards tab. On the left side of the panel is the location of the card followed by the timestamp of when the data was reported. The next few columns display the most recent temperature of the sensors on the card and, in most cases, information about the power. In Navigator the temperatures are always color coded, with blue being the coolest temperatures and red representing the warmest temperatures.

Tip: You can adjust the color coding in the Environmental Queries pages of the Navigator to suit your environment. In the `/bgsys/local/etc/navigator-config.xml` file, add the following Environmentals container (shown with default values):

```
<environmentals>
  <min-temp>25.0</min-temp>
  <max-temp>50.0</max-temp>
</environmentals>
```

Any temperatures that are below the `min-temp` value display in blue. Temperatures above the `max-temp` value show in red. Values between the two settings are shades of blue, green, and red. You can reset the values and save your changes. The new settings take effect the next time that you start the Navigator server. These settings affect only the colors that display in the Navigator.



Environmental Queries

Bulk Power Modules | Link Card Power | Node Card Power | Service Card Power

Clock Cards | Fans | Link Cards | Link Chips | Node Cards | Nodes | Service Cards

▶ Filter Options

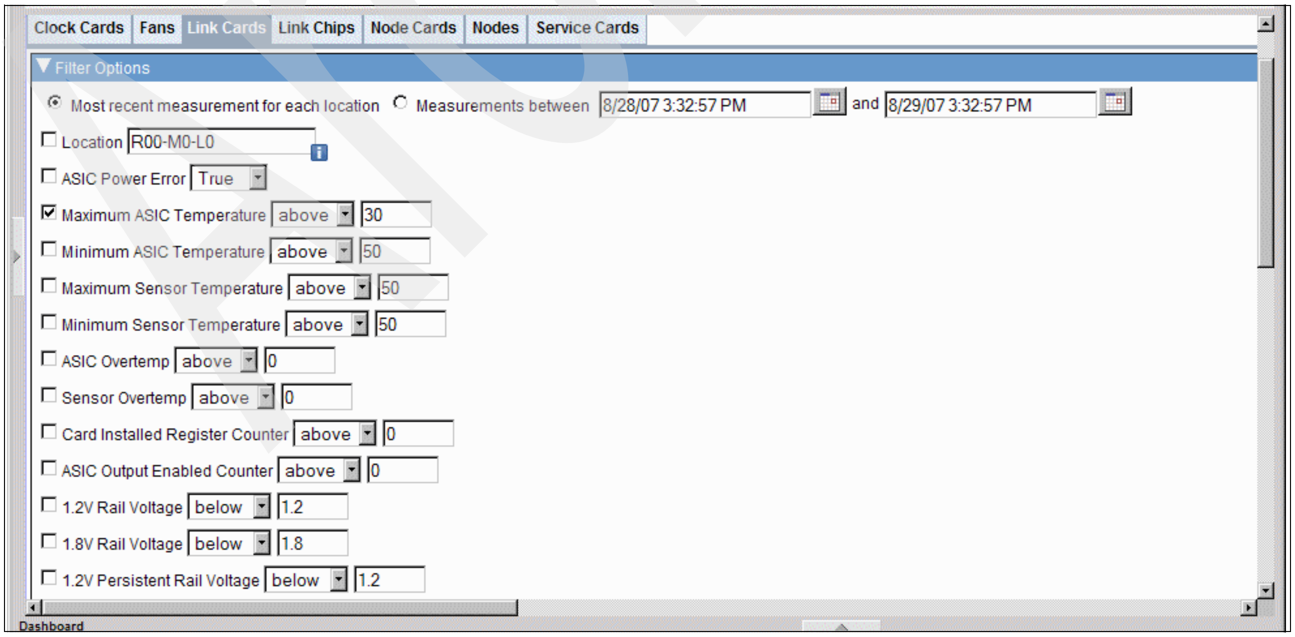
Location	Time	ASIC Power Error	Max ASIC Temp	Min ASIC Temp	Max Sensor Temp	Min Sensor Temp	ASIC Overtemp	Sensor Overtemp	Card Installed
R00-M0-L0	8/29/07 3:52:52 PM	False	33	23	25	22	0	0	
R00-M0-L1	8/29/07 3:52:52 PM	False	22	16	21	17	0	0	
R00-M0-L2	8/29/07 3:52:52 PM	False	26	20	25	21	0	0	
R00-M0-L3	8/29/07 3:52:52 PM	False	31	19	22	17	0	0	
R00-M1-L0	8/29/07 3:52:52 PM	False	33	19	25	21	0	0	
R00-M1-L1	8/29/07 3:52:52 PM	False	33	17	24	18	0	0	
R00-M1-L2	8/29/07 3:52:52 PM	False	22	19	25	21	0	0	
R00-M1-L3	8/29/07 3:52:52 PM	False	20	16	21	18	0	0	
R01-M0-L0	8/29/07 3:52:52 PM	False	33	18	25	18	0	0	
R01-M0-L1	8/29/07 3:52:52 PM	False	31	17	21	16	0	0	
R01-M0-L2	8/29/07 3:52:52 PM	False	39	20	25	21	0	0	
R01-M0-L3	8/29/07 3:52:52 PM	False	21	17	21	17	0	0	
R01-M1-L0	8/29/07 3:52:52 PM	False	29	18	22	18	0	0	
R01-M1-L1	8/29/07 3:52:52 PM	False	33	18	21	17	0	0	
R01-M1-L2	8/29/07 3:52:52 PM	False	27	18	23	19	0	0	

Dashboard

Figure 5-4   Environmental Queries, Link Card tab

Many times, the amount of data that is returned can seem overwhelming. By selecting one or more of the various column heads, you can refine the results returned by Environmental Queries. Click **Filter Options** just below the hardware tabs to open the filter window. Figure 5-5 shows the amount of data reduced by selecting **Most recent measurement for each location** and only the Link cards that have a *Max ASIC Temp* over 30 degrees.

Note: When you go to the Nodes tab, rather than collecting the environmental data for all of the node cards, data is returned only for R00-M0-N00-J04 to reduce the response time. After the initial page is loaded, you can use the Filter Options to alter the results.



Clock Cards | Fans | Link Cards | Link Chips | Node Cards | Nodes | Service Cards

▼ Filter Options

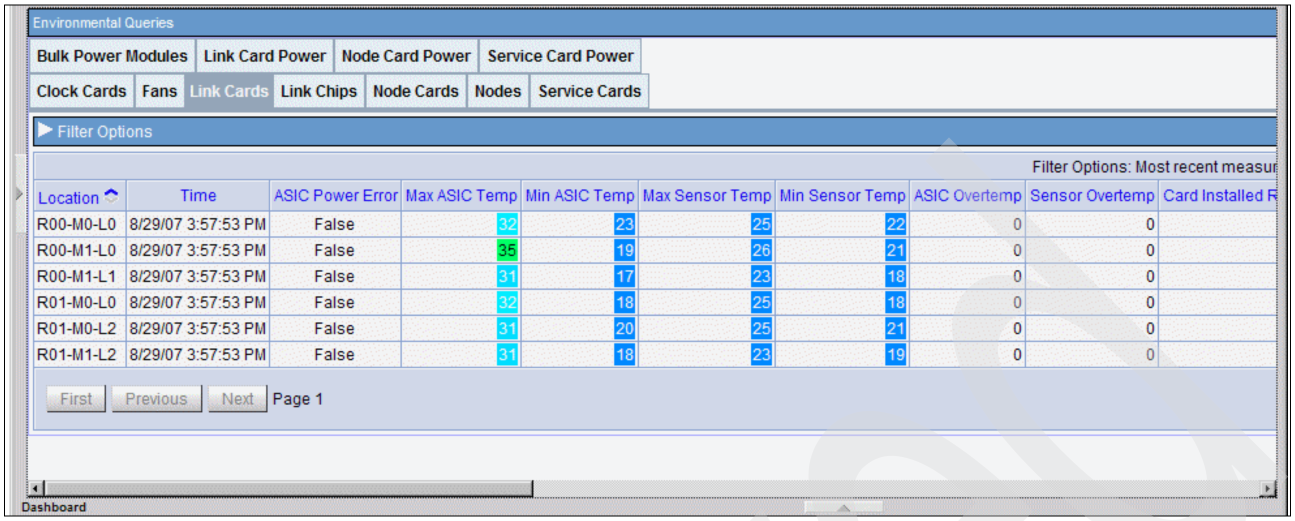
[✓ Most recent measurement for each location] Measurements between 8/28/07 3:32:57 PM and 8/29/07 3:32:57 PM

☐	Location		R00-M0-L0
☐	ASIC Power Error	True	
☑	Maximum ASIC Temperature	above	30
☐	Minimum ASIC Temperature	above	50
☐	Maximum Sensor Temperature	above	50
☐	Minimum Sensor Temperature	above	50
☐	ASIC Overtemp	above	0
☐	Sensor Overtemp	above	0
☐	Card Installed Register Counter	above	0
☐	ASIC Output Enabled Counter	above	0
☐	1.2V Rail Voltage	below	1.2
☐	1.8V Rail Voltage	below	1.8
☐	1.2V Persistent Rail Voltage	below	1.2

Dashboard

Figure 5-5   Environmental Queries Filter Options

As shown in Figure 5-6, after clicking **Apply Filters**, the query returns only six link cards.



Filter Options: Most recent measur

Location	Time	ASIC Power Error	Max ASIC Temp	Min ASIC Temp	Max Sensor Temp	Min Sensor Temp	ASIC Overtemp	Sensor Overtemp	Card Installed R
R00-M0-L0	8/29/07 3:57:53 PM	False	32	23	25	22	0	0	
R00-M1-L0	8/29/07 3:57:53 PM	False	35	19	26	21	0	0	
R00-M1-L1	8/29/07 3:57:53 PM	False	31	17	23	18	0	0	
R01-M0-L0	8/29/07 3:57:53 PM	False	32	18	25	18	0	0	
R01-M0-L2	8/29/07 3:57:53 PM	False	31	20	25	21	0	0	
R01-M1-L2	8/29/07 3:57:53 PM	False	31	18	23	19	0	0	

Figure 5-6 Filter results

Running your queries through the Navigator in the Environmental Monitor is much simpler than submitting long strings on the command line, and the results are far more readable.

6

High Throughput Computing

High Throughput Computing (HTC) first became available in Blue Gene/P with V1R2M0. Contrary to the traditional High Performance Computing (HPC) model, HTC might use each of the compute nodes in a partition to run a different executable. HTC mode on Blue Gene/P is integrated completely into the Midplane Management Control System (MMCS), the control system of Blue Gene/P and, thus, is fully aware of all HTC jobs. These jobs are represented in the Blue Gene database when they are running. You can view them either from the MMCS console on the service node or by using the Blue Gene Navigator from a browser. When jobs complete, information about the job is stored in the job history database table.

This chapter discuss how HTC works, the security features of HTC, and how to set up HTC.

6.1 How HTC works

In the Blue Gene/P version of HTC, jobs enter the system through submit clients that are running on a submit node. The *submit node* is usually a front end node but can also be the service node. The submit client connects to a *software multiplexer* (mux) that is running on the same node. The mux is responsible for forwarding communications from each of its submit clients to the service node's Submit Server daemon.

The primary role of the Submit Server daemon is to manage system resources. When a partition boots in HTC mode, the available partition resource states are managed in the Submit Server daemon's process memory so that a fast lookup can be done. After the Submit Server daemon accepts and authenticates the incoming connection, the Submit Server daemon determines if the necessary resources are available to complete the submit client's request. Based on the availability, the request is accepted or denied. If the request is accepted, the Submit Server daemon returns the allocated resource back to the submit client.

Figure 6-1 shows an overview of the HTC architecture.

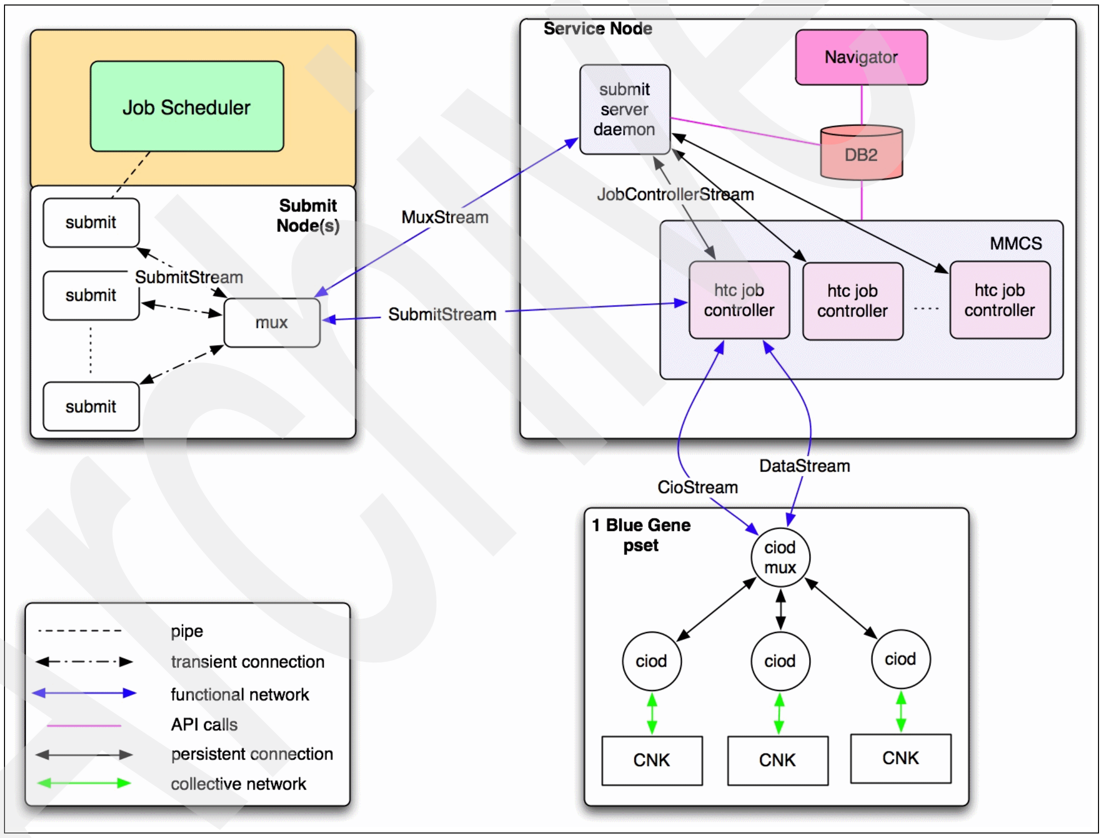


Figure 6-1 HTC Overview

For each HTC partition that is booted, the MMCS Server starts an associated HTC *job controller*. The job controller connects to the Submit Server daemon and reports its availability and a list of managed resources. In return, the Submit Server daemon sends a list of active submit muxes and details about how to connect to them. The connection between the job controller and the Submit Server daemon is made on a configurable port and remains open as long as the partition is allocated.

The job controller also opens a connection to each Control I/O Daemon (CIOD) that is running in the partition or a connection to the CIOD mux, which represents the collection of all the CIODs running on the I/O node. These connections are used to handle both job control messages (start, stop, and so forth) and job data (stdin, stdout, and stderr).

When the submit client receives an allocated resource back from the Submit Server daemon, it sends the start (run) job request to the HTC job controller that owns the compute resource. All submit requests flow through a mux that has a persistent connection to the Submit Server daemon and every HTC job controller. Outside of allocating compute resources, which is done through the Submit Server daemon, all job management requests (starting, stopping, and so forth) flow between the mux and the HTC job controller. Additionally, all data flow (stdin, stdout, and stderr) travels between the mux and the HTC job controller.

6.2 Security

In this section we discuss the following security features in the HTC design:

- Submit client to submit mux
- Submit mux to the Submit Server
- Submit mux to the job controller

6.2.1 Submit client to submit mux

The submit mux listens on *localhost* only for connections from submit clients. By default, there is no authentication between the submit mux and submit client. Normally, it is expected that anyone with a login to a front end node is authorized automatically to use the **`submit`** command.

When an extra layer of security is needed, you can use the `--authclients` argument with the **`submit_mux start`** command to require a handshake between the submit mux and submit client. By default, this argument is disabled to provide the highest possible job throughput available. When this argument is enabled, the submit client needs to read the challenge passphrase from the `/bgsys/local/etc/submit_mux.cfg` file. To do so, give the **`submit`** command attributes with user bgpadmin, which can be enabled with following commands:

```
chown bgpadmin:bgpadmin /bgsys/drivers/ppcfloor/bin/submit
chmod +s /bgsys/drivers/ppcfloor/bin/submit
```

Enabling this argument provides the **`submit`** command the same level of security as mpirun.

Tip: If you want to prevent specific users or groups from accessing the **`submit`** command, use the standard Linux file access control mechanisms.

6.2.2 Submit mux to Submit Server authentication

Submit muxes that attempt to connect to the Submit Server daemon running on a service node must use a challenge and response authentication protocol. This protocol is modeled after the mpirun to mpirun daemon method of authentication done on Blue Gene/P. After the mux has authenticated to the Submit Server daemon, it becomes a trusted service provider for submit clients.

The challenge password that the submit mux uses on the submit node is stored in the `/bgsys/local/etc/submit_mux.cfg` file. The owner of the file is bgpadmin, and the group is bgpadmin. Both the owner and group have read/write access, whereas all others have no access.

The challenge password that the Submit Server daemon uses on the service node is stored in the /bgsys/local/etc/submit_server.cfg file. The owner of the file is bgpadmin and the group is bgpadmin. Both the owner and group have read/write access, whereas all others have no access.

The challenge password must match in both /bgsys/local/etc/submit_mux.cfg and /bgsys/local/etc/submit_server.cfg for the submit mux to authenticate to the Submit Server daemon. Both the Submit Server daemon and the submit mux daemon must be restarted for password changes to take effect.

6.2.3 Submit mux to HTC job controller

An HTC job controller connects to each submit mux after registering itself with the Submit Server daemon. Authentication occurs according to the following procedure:

1. The submit mux generates a random string and sends this value to the Submit Server daemon as part of the registration process.
2. The Submit Server daemon sends this random string to each registering job controller.
3. The HTC job controller sends this random string to the submit mux when it connects.
4. The submit mux verifies that the random string sent by the HTC job controller matches the string generated in step 1.

6.3 Setting up HTC

This section discuss the necessary steps to enable HTC mode.

6.3.1 Starting the Submit Server on the service node

You can find a template of the startsubmitserver file in the /bgsys/drivers/ppcfloor/bin directory. Log in as bgpadmin, and copy this file to the local configuration using the following command:

```
cp /bgsys/drivers/ppcfloor/bin/startsubmitserver.tpl
/bgsys/local/etc/startsubmitserver
```

After you copy the file, verify that both the owner and group are set to bgpadmin. Verify that the owner and group both have read/write/execute authority to the file and that the public has no authority to access the file. You can set ownership and authorities with the following commands:

```
chown bgpadmin:bgpadmin /bgsys/local/etc/startsubmitserver
chmod 770 /bgsys/local/etc/startsubmitserver
```

6.3.2 Updating the production bgpmaster.init file

Starting with V1R2M0, the template file, /bgsys/drivers/ppcfloor/bin/bgpmaster.init.tpl, is updated to start the Submit Server daemon. (Example 3-1 on page 82 shows contents of the bgpmaster.init file.) You need to apply these changes to the production bgpmaster.init file.

If you have *not* made any local configuration changes to the `/bgsys/local/etc/bgpmaster.init` file, you can copy the template file to your local configuration using the following command:

```
cp /bgsys/drivers/ppcfloor/bin/bgpmaster.init.tpl
/bgsys/local/etc/bgpmaster.init
```

If you *have* made changes to the local version of the file, you can edit the `/bgsys/local/etc/bgpmaster.init` file and add the lines that pertain to the Submit Server.

The authorities on the bgpmaster.init file are 644, with both the owner and group set to bgpadmin.

6.3.3 Creating the submit_mux.cfg file

The `submit_mux.cfg` file contains configuration settings that are used by the `.init` scripts to start the submit mux on submit nodes automatically. Typically, a submit node is a front end node where the **`submit`** command is executed. The configuration file also contains the challenge password that is used to authenticate a submit mux with the Submit Server daemon running on the service node. You must create the `submit_mux.cfg` file in the `/etc` directory with the proper file authorities and configuration values set.

You can find the template for `submit_mux.cfg` at `/bgsys/drivers/ppcfloor/bin/submit_mux.cfg.tpl`, and the Blue Gene System Administrator can copy this file using the following command:

```
cp /bgsys/drivers/ppcfloor/bin/submit_mux.cfg.tpl
/bgsys/local/etc/submit_mux.cfg
```

After the file is copied, the correct file authorities must be applied. The owner of the file is bgpadmin with read/write authority, group authority is bgpadmin with read/write authority, and all others have no access to the file. The correct file authority settings are very important because the challenge password is contained in the file. To set the ownership and privileges on the file, use these commands:

```
chown bgpadmin:bgpadmin /bgsys/local/etc/submit_mux.cfg
chmod 660 /bgsys/local/etc/submit_mux.cfg
```

Customizing the /bgsys/local/etc/submit_mux.cfg configuration file

Typically, you need to change only two settings in the `submit_mux.cfg` file:

- Set `SUBMIT_SERVER_HOSTNAME` to your Submit Server (service node) host name (for example, `bgp.yourSite.com`).
- Set `CHALLENGE_PASSWORD` to the challenge password that is configured in the `/bgsys/local/etc/submit_server.cfg` file on the service node where the Submit Server is running. The values for `CHALLENGE_PASSWORD` in `/bgsys/local/etc/submit_mux.cfg` (file on `Submit Node/FEN`) and `/bgsys/local/etc/submit_server.cfg` (file on service node) must match.

Starting the submit mux daemon on the submit node

To start the submit mux daemons on the submit node:

1. Copy the `.init` script to `/etc/init.d`.
2. Install the submit mux daemon to start automatically at the default run levels.
3. Start the script manually.

You can find the template for the `ibm.com-submit_mux` init script at `/bgsys/drivers/ppcfloor/bin/ibm.com-submit_mux.tpl`. The Blue Gene System Administrator can copy this file using the following command:

```
cp /bgsys/drivers/ppcfloor/bin/ibm.com-submit_mux.tpl
/etc/init.d/ibm.com-submit_mux
```

After the file is copied, apply the correct file authorities. The owner of the file is bgpadmin with read/write/execute authority, group authority is bgpadmin with read/write/execute authority, and all others have no access to the file. Use the following commands:

```
chown bgpadmin:bgpadmin /etc/init.d/ibm.com-submit_mux
chmod 770 /etc/init.d/ibm.com-submit_mux
```

To install the submit mux daemon, issue the following command:

```
/usr/lib/lsb/install_initd -v ibm.com-submit_mux
```

Then start the daemon using this command:

```
/etc/init.d/ibm.com-submit_mux start
```

7

RAS messages

The Reliability, Availability, and Serviceability (RAS) reporting structure has changed considerably from Blue Gene/L. Rather than a text based format, Blue Gene/P uses a message ID format that provides a consistent interface to analyze messages. This chapter discusses this interface.

7.1 Blue Gene/P RAS

RAS messages have evolved from the text based messages used in Blue Gene/L to messages that have a defined structure in Blue Gene/P. Basic information, such as a message ID, severity, location, and text related to the event are available at first glance on the *RAS Event Log* page in Navigator. You can obtain additional information, such as the component, raw data, and service suggestions by clicking the additional information icon next to the message. Formatting the messages in this manner improves the usability by providing uniformity in the delivery of RAS events.

The Blue Gene/P version of Diagnostics also uses the RAS infrastructure to report errors and progress. This feature provides a central location that administrators and support personnel can use when looking for events.

Blue Gene/P also provides the ability for users to define RAS events from within their program. There are 40 pre-created, user definable RAS events that can be used to log messages in the event log.

You can preview all of the RAS messages with the basic information included by clicking the **RAS Message Types** link in the Navigator. You can also find this information in the `/bgsys/drivers/ppcfloor/doc/RasEventBook.htm` file, which you can view with a browser.

7.2 RAS event message flow

RAS events can be generated from several sources:

- The kernel can create an event and send it to the Machine Controller (MC) using the mailbox protocol.
- The card controllers in the MC can also create RAS events.
- Diagnostics generate RAS events to report test results.

Events coming from the kernel or hardware are read and interpreted by the MC and passed to a mcServer listener. Messages created by Diagnostics are sent directly to the mcServer listener. The mcServer relays the events to a registered listener on the MMCS server. The MMCS server then logs the RAS event to the database, at which time it can be viewed from the Navigator or by querying the database directly.

Figure 7-1 shows the path that the RAS messages follow.

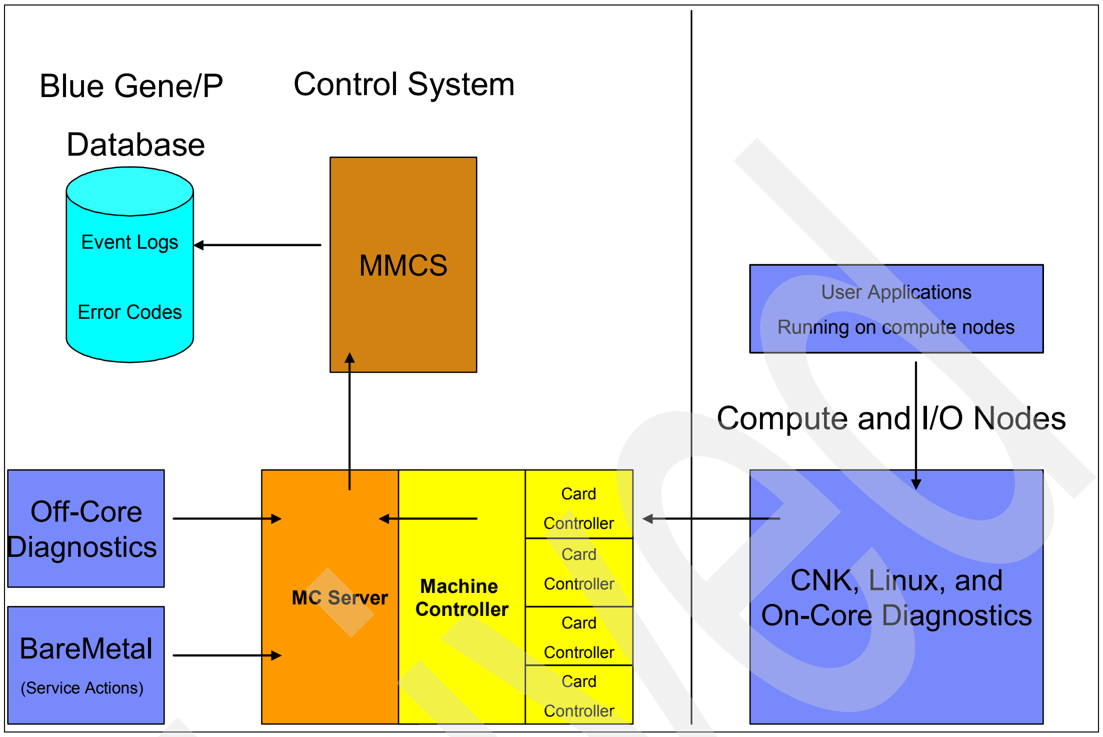


Figure 7-1 RAS Event flow

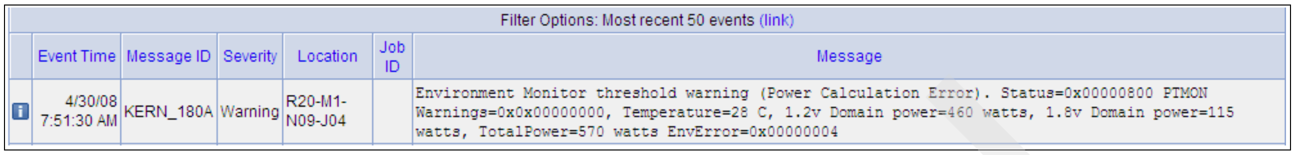
7.3 Anatomy of a RAS message

Each RAS event message consists of several components:

- Message ID
- Component
- Subcomponent
- Error Code
- Severity
- Location
- Message
- Detailed description
- Recommended Service Action
- Replacement Policy
- Diagnostics recommendations

Depending on the context in which you view the message, you might see other information, such as the date and time of the event, the number of occurrences, or a job ID.

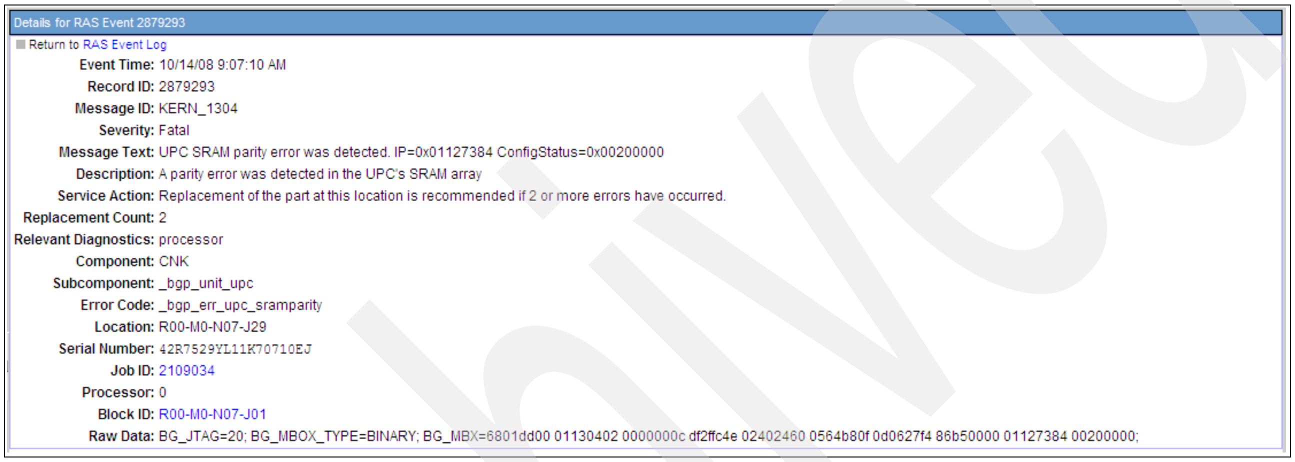
Figure 7-2 shows that many of the message components are column headings in Navigator's RAS event log. In the Navigator environment you can sort the results by each of the column heads by clicking the heading itself.



Filter Options: Most recent 50 events (link)						
	Event Time	Message ID	Severity	Location	Job ID	Message
🛈	4/30/08 7:51:30 AM	KERN_180A	Warning	R20-M1-N09-J04		Environment Monitor threshold warning (Power Calculation Error). Status=0x00000800 PTMON Warnings=0x0x00000000, Temperature=28 C, 1.2v Domain power=460 watts, 1.8v Domain power=115 watts, TotalPower=570 watts EnvError=0x00000004

Figure 7-2 Column heads

Notice the *i* icon in the left most column of the page (Figure 7-2). Clicking this icon link opens the details for that specific error. Figure 7-3 shows the message details page after clicking the information icon.



Details for RAS Event 2879293

Return to RAS Event Log

Event Time:	10/14/08 9:07:10 AM
Record ID:	2879293
Message ID:	KERN_1304
Severity:	Fatal
Message Text:	UPC SRAM parity error was detected. IP=0x01127384 ConfigStatus=0x00200000
Description:	A parity error was detected in the UPC's SRAM array
Service Action:	Replacement of the part at this location is recommended if 2 or more errors have occurred.
Replacement Count:	2
Relevant Diagnostics:	processor
Component:	CNK
Subcomponent:	_bgp_unit_upc
Error Code:	_bgp_err_upc_sramparity
Location:	R00-M0-N07-J29
Serial Number:	42R7529YL11K70710EJ
Job ID:	2109034
Processor:	0
Block ID:	R00-M0-N07-J01
Raw Data:	BG_JTAG=20; BG_MBOX_TYPE=BINARY; BG_MBX=6801dd00 01130402 0000000c df2ffc4e 02402460 0564b80f 0d0627f4 86b50000 01127384 00200000;

Figure 7-3 Message details

In the following sections, we describe the various components in more detail.

Message ID

Each message includes a *Message ID* that is made up of four letters and an underscore followed by four digits. The letters indicate the source of the message. For example, a `KERN_`*`xxxx`* originated from the kernel. The following components can generate a RAS message and the associated ID:

- Compute Node Kernel (`KERN_`*`xxxx`*)
- mcServer (`MCTL_`*`xxxx`*)
- Diagnostics (`DIAG_`*`xxxx`*)
- Machine Controller (`MCTL_`*`xxxx`*)
- BareMetal™ (`BRMT_`*`xxxx`*)
- MMCS (`MMCS_`*`xxxx`*)
- Card Controller (`CARD_`*`xxxx`*)
- Application (`APPL_`*`xxxx`*)
- User-defined (`USER_`*`xxxx`*)

Subcomponent and error codes

For each of the components, *subcomponents* indicate the functional area that originated the message. Each subcomponent also has *error codes* that indicate the specific condition of the RAS event. These subcomponents and error codes are stored separately in the RAS database for each RAS message. They are also combined with the component to make up the full message ID. The four digits that come after the component in the message ID are determined by the subcomponent (two digits) and the error code (two digits). For example, the RAS event for Component MMCS, Subcomponent HARDWARE_MONITOR, and Error Code BULK_POWER_WARNING has a Message ID of MMCS_0206.

Severity

The message also includes the *severity* of the event:

`FATAL`	Designates a severe error event that presumably will lead the application to fail or abort.
`ERROR`	Designates error events that might still allow the application to continue running, such as failure of a redundant component.
`WARN`	Designates potentially harmful situations such exceeding a soft error threshold.
`INFO`	Designates informational messages that highlight the progress of system software.
`TRACE`	Designates fine-grained informational events that are useful to debug code.
`DEBUG`	Designates more fine-grained informational events that are most useful to debug code.
`UNKNOWN`	The severity is not known. UNKNOWN messages are unexpected and typically represent a code or build bug.

Message

The *message* contains a brief overview of the condition. Depending on the message, it might provide the component, location, or type of error. The message can include substitution variables such as a return code, torus coordinates, error count, and so forth. Therefore, the message text can vary for the same message ID.

Detailed event description

The *event description* gives a more general description of the event. Typically, in this area, you see events that lead to an additional message, such as environmentals that are being read or that a specific diagnostic test is running. The description might also suggest the source of the problem. For example, the following message came from the description area of message MMCS_0207:

```
While reading environmental data from the hardware, MMCS detected an error in a
card's power module. The error can be related to temperature, current, voltage,
or a general fault with the module.
```

Recommended Service Action

Based on the event, the message might also include a Service Action *recommendation*. Some errors might always result in the replacement of a compute node. If that is the case, the message indicates that it will resolve the problem. In other cases, you might be instructed to run diagnostics or to verify a reading on the Environmental Queries page to determine whether the event is a transient issue. Many of the recommendations to run diagnostics include the relevant test buckets that you need to select.

The action might specify a replacement policy based on the number of errors observed within a given period. In Example 7-1, the RAS details include a Replacement Count (threshold) and Replacement Period. The service action recommends the part that needs to be replaced if the count of errors exceed the threshold within a given period. The service action, replacement recommendations and relevant diagnostic test buckets are highlighted in bold in this example.

Example 7-1 RAS event details

```
Event Time: 10/8/08 10:48:44 PM
Record ID: 2816161
Message ID: KERN_0802
Severity: Warning
Message Text: DDR correctable single symbol error(s): DDR Controller 1, failing SDRAM address
0x03ee4ef20, BPC pin JM194, transfer 1, bit 41, BPC module pin AA20, compute trace
MEMORY1DATA50, DRAM chip U39, DRAM pin D7.

Description: The DDR controller detected and corrected a single symbol error. If more than 5
correctables are detected on the same wire, CNS will attempt to activate redundant bit steering
(RBS) to avoid the intermittent wire to/location on the DRAM. If RBS is activated, there will be
a _bgp_err_ddr_rbs_activated RAS message.

Service Action: Replacement of the part at this location is recommended if 120 or more errors
have occurred in a period of 1 hour or less.

Replacement Count: 120
Replacement Period: 1 hour
Relevant Diagnostics: memory
Component: CNK
Subcomponent: _bgp_unit_ddr
Error Code: _bgp_err_ddr_single_symbol_error
Location: R30-M1-N01-J21
Serial Number: 42R7523YL12K708205H
Processor: 0
Block ID: SYSS_R30-M1
Raw Data: BG_JTAG=8; BG_MBOX_TYPE=BINARY; BG_MBX=69845500 01080207 0000000e ef495b6c 02402460
0564b80f 090d2777 88a50000 00000f00 0000000d 00000001 a8000000 00000008 3ee4ef20 00000001;
```

Other information

Some of the data displayed is dependent on the link that you are viewing. For example, if you are looking at events in the RAS Event Log, you can see the Event Time and possibly a Job ID that is associated with a specific error. If you are viewing the events from the RAS Message Types, you can see how many occurrences of each type of event is logged on the system during the time period selected in the Filter Options.

7.4 System RAS policies

The control system kills the job and frees the block for KERNEL FATAL RAS Events. These events have a message ID of `KERN_XXXX` and a severity of `FATAL`.

Although the event is fatal to the job, the condition might be *transient* or *soft*, which means that hardware failures occur and cause an application to fail but that the application can be restarted on the same hardware without hardware repair. Soft errors are most likely radiation induced and are a natural consequence of the impact of terrestrial and cosmic radiation sources, as well as radioactive impurities in the semiconductor and packaging materials close to the CMOS circuits. Blue Gene/P is designed to reduce the incidence of radiation.

Other failures are *hard*. Hard failures mean that the failed part permanently exhibits the failure mode and that the hardware must be replaced to remedy the failure.

An RAS event can include a Control Action that indicates that the condition is likely hard and warrants marking the hardware in error. Marking hardware that is in error helps job schedulers such as Load Leveler avoid dispatching jobs to failing hardware. The possible Control Actions include:

`NC_IN_ERROR`	For node card (and nodes), this action marks the node card in Error in the database and prevents any blocks that use the node card from booting.
`LC_IN_ERROR`	For link card (and chips), this action marks a link card in Error in the database and prevents any blocks that use the link card from booting.
`CABLE_IN_ERROR`	For Cable (and link ports), this action marks a cable in Error in the database and prevents any blocks that use the cable from booting.

Service actions, which we discuss in Chapter 11, “Service Actions” on page 199, are required to return the hardware to an active state.

The control system Environmental Monitoring can log `WARNING`, `ERROR`, and `FATAL` events. The Environmental Monitoring `FATAL` errors are associated with failed bulk power modules. These failures can lead to hardware marked in error in the database, as shown in the following scenarios:

- If 1 of 9 bulk power modules in a rack are faulted, an RAS Event MMCS_0205 (SEV=ERROR) is issued.
- If 2 of 9 bulk power modules in a rack are faulted, an RAS Event MMCS_020B (SEV=FATAL) is issued.   The top midplane’s node cards are marked in Error in the database and associated jobs are killed.
- If 3 or more of 9 bulk power modules in a rack are faulted, an RAS event MMCS_020B (SEV=FATAL) is issued. The entire rack’s node cards are marked in Error in the database
- If 2 or more service card power modules on the same service card and in same voltage domain are faulted, an RAS Event MMCS_020A (SEV=FATAL) is issued. The midplane containing the service card has its node cards marked in Error in the database and associated jobs are killed. The service card power module domains are as follows:
  - Modules P0 and P1 are in the 1.8 volt domain
  - Modules P2, P3, and P6 are in the 3.5 volt domain
  - Modules P4 and P5 are in the 5.0 volt domain

7.5 Tailoring RAS Events

The control system has a mechanism to alter RAS events. These alterations are typically guided by IBM Support as needed to circumvent a problem while waiting for a bug fix. The mechanism reads a set of change specifications from the `ras_environment_filter.xml` file in the `/bgsys/local/etc` directory during server start. If the file is missing from that location, the mechanism reads the file from its original installation location at `/bgsys/drivers/ppcfloor/bin`.

The change specifications indicate the message ID of interest and list the RAS event item to change based on its key name. Example 7-2 shows how to alter the outcome of a RAS event.

Example 7-2 Sample ras_environment_filter.xml file

```
<BgRasEnvironments>
   <BgRasEnvironment environment="PROD">
      <!-- Change the severity of an event -->
      <RasEventChangeSpec id="KERN_XXXX" key="BG_SEV" value="FATAL"/>
      <!-- Change the control action - this action marks the node in Error -->
      <RasEventChangeSpec id="KERN_XXXX" key="BG_CTL_ACT" value="NC_IN_ERROR"/>
      <!-- Change the control action to not mark the node in Error -->
      <RasEventChangeSpec id="KERN_XXXX" key="BG_CTL_ACT" value="NONE"/>
   </BgRasEnvironment>
</BgRasEnvironments>
```

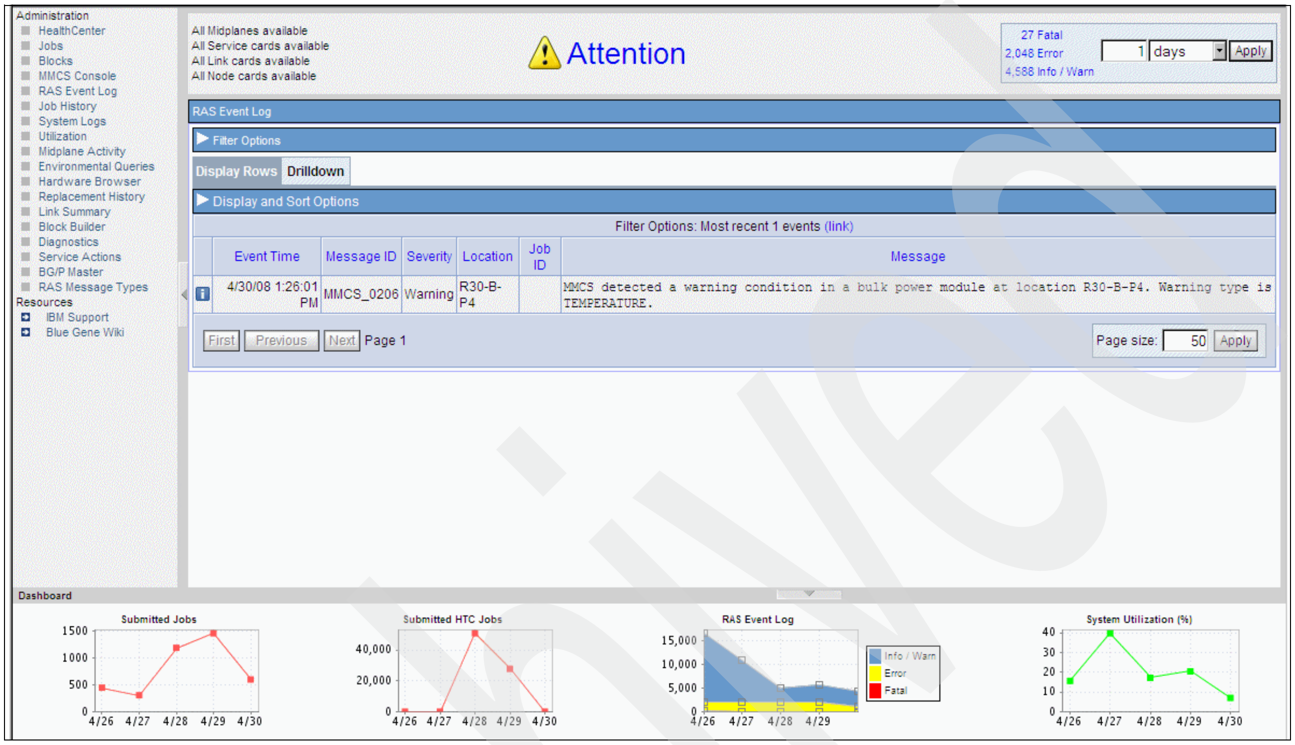
The first `RasEventChangeSpec` shows how to change an RAS event severity for message ID `KERN_`*`XXXX`* (where *`XXXX`* is a valid subcomponent and error code). The second `RasEventChangeSpec` shows how to add a control action to mark the hardware in error. Marking the hardware that is in error makes the hardware unavailable until a service action is performed. The third `RasEventChangeSpec` shows how to prevent the hardware from being marked in error. The control action is changed to `NONE`.

The control system logs the active change specifications during startup and displays the following message:

```
RAS EventFilter active for KERN_XXXX, set BG_CTL_ACT=NC_IN_ERROR
```

7.6 Navigator interface to the RAS database

Clicking **RAS Event Log** in Blue Gene Navigator opens the page shown in Figure 7-4. This interface provides many features that simplify locating, quantifying, and analyzing RAS messages.



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

Filter Options: Most recent 1 events (link)

	Event Time	Message ID	Severity	Location	Job ID	Message
i	4/30/08 1:26:01 PM	MMCS_0206	Warning	R30-B-P4		MMCS detected a warning condition in a bulk power module at location R30-B-P4. Warning type is TEMPERATURE.

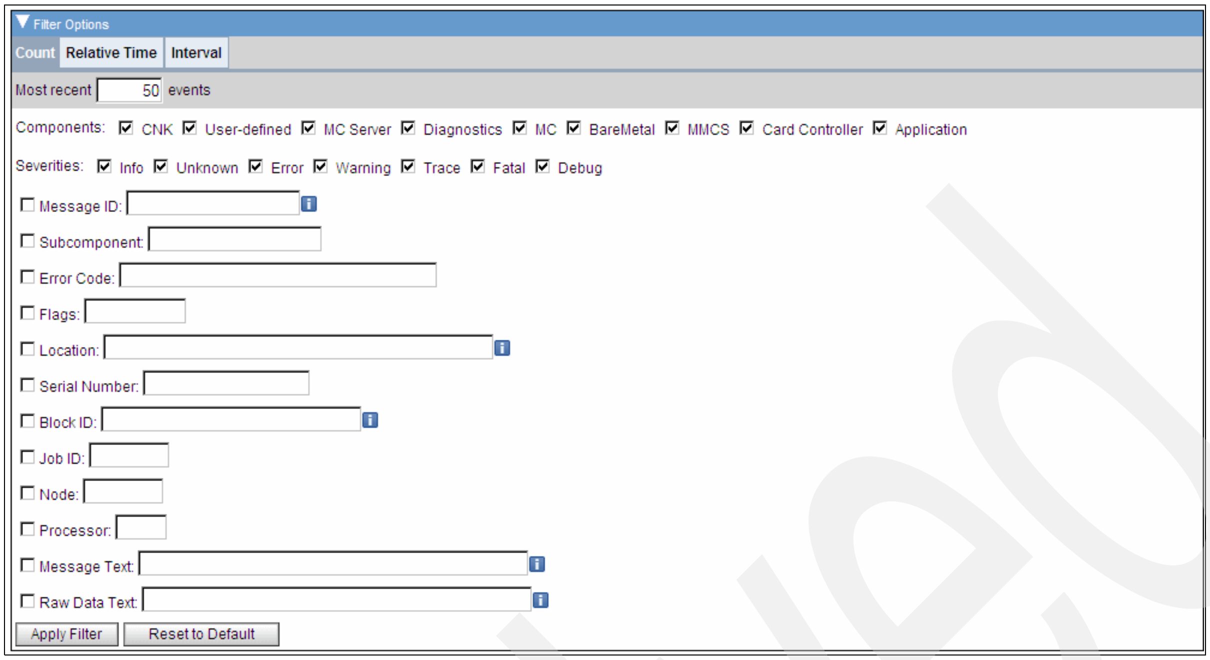
Figure 7-4 RAS Event Log

Filter Options

Figure 7-5 shows the available options when you expand the Filter Options link on the RAS Event Log page. The tabs directly under the Filter Options heading allow you to limit the data returned by *Count*, *Relative Time*, or *Interval*. You input the following information about each of the tabs:

Count	Supply the maximum number of entries that you want to return. The default is 50.
Relative Time	Provide the number of seconds, minutes, hours, or days that you want to go back in time to search for messages. The default is 30 minutes.
Interval	Allows you to enter a start and stop time as limits to the search. The default is the preceding 24-hour period.

The common area for each of the tabs allows you to select the various Components and Severities that you can use to filter the data that is returned. You can further narrow the results by specifying search criteria from the extended filter list such as Message ID, Subcomponent, Error Code, and so on. Add the criteria to your search by selecting the box next to the filter and then supplying a value. When you hover over the *i* icon (next to the input area), you see information such as string limits, wild cards, and search options.



Most recent 50 events

Components: ☑ CNK ☑ User-defined ☑ MC Server ☑ Diagnostics ☑ MC ☑ BareMetal ☑ MMCS ☑ Card Controller ☑ Application

Severities: ☑ Info ☑ Unknown ☑ Error ☑ Warning ☑ Trace ☑ Fatal ☑ Debug

☐ Message ID:

☐ Subcomponent:

☐ Error Code:

☐ Flags:

☐ Location:

☐ Serial Number:

☐ Block ID:

☐ Job ID:

☐ Node:

☐ Processor:

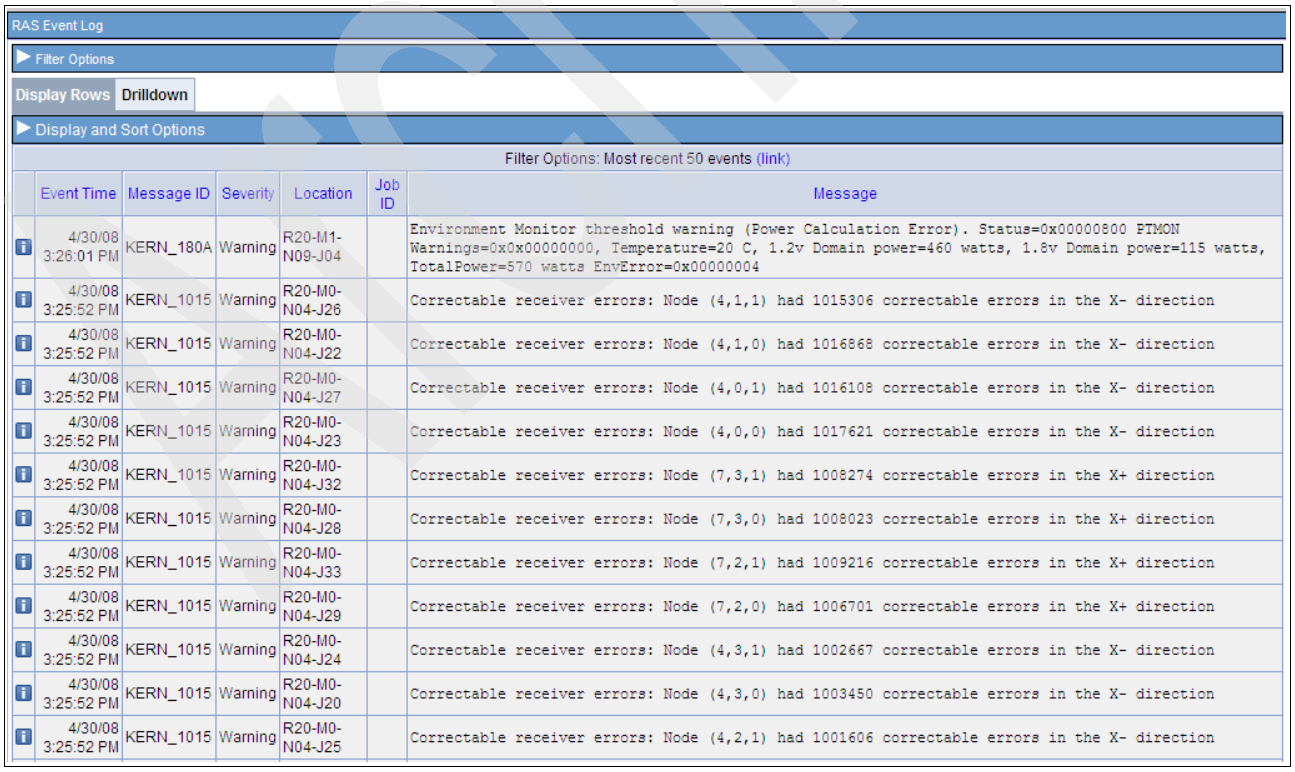
☐ Message Text:

☐ Raw Data Text:

Figure 7-5 Filter options

Display Rows versus Drilldown

The default view of RAS Event Log is *Display Rows* (shown in Figure 7-6). In this view, you can sort each of the columns in ascending or descending order by clicking the column heading. The information icon to the left opens a window with more details about this specific instance of the error.

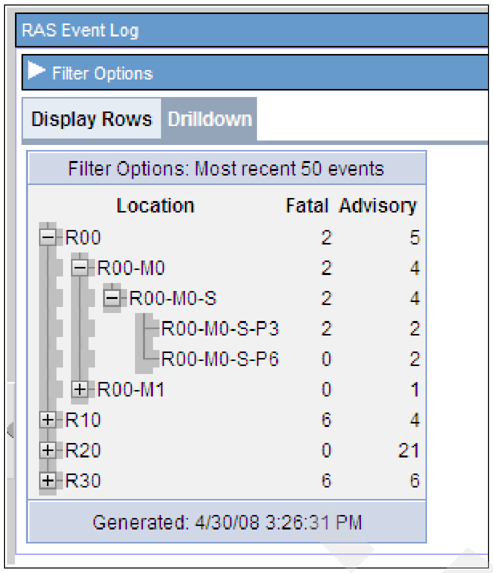


Filter Options: Most recent 50 events (link)

	Event Time	Message ID	Severity	Location	Job ID	Message
ℹ	4/30/08 3:26:01 PM	KERN_180A	Warning	R20-M1-N09-J04		Environment Monitor threshold warning (Power Calculation Error). Status=0x00000800 PTMON Warnings=0x0x00000000, Temperature=20 C, 1.2v Domain power=460 watts, 1.8v Domain power=115 watts, TotalPower=570 watts EnvError=0x00000004
ℹ	4/30/08 3:25:52 PM	KERN_1015	Warning	R20-M0-N04-J26		Correctable receiver errors: Node (4,1,1) had 1015306 correctable errors in the X- direction
ℹ	4/30/08 3:25:52 PM	KERN_1015	Warning	R20-M0-N04-J22		Correctable receiver errors: Node (4,1,0) had 1016868 correctable errors in the X- direction
ℹ	4/30/08 3:25:52 PM	KERN_1015	Warning	R20-M0-N04-J27		Correctable receiver errors: Node (4,0,1) had 1016108 correctable errors in the X- direction
ℹ	4/30/08 3:25:52 PM	KERN_1015	Warning	R20-M0-N04-J23		Correctable receiver errors: Node (4,0,0) had 1017621 correctable errors in the X- direction
ℹ	4/30/08 3:25:52 PM	KERN_1015	Warning	R20-M0-N04-J32		Correctable receiver errors: Node (7,3,1) had 1008274 correctable errors in the X+ direction
ℹ	4/30/08 3:25:52 PM	KERN_1015	Warning	R20-M0-N04-J28		Correctable receiver errors: Node (7,3,0) had 1008023 correctable errors in the X+ direction
ℹ	4/30/08 3:25:52 PM	KERN_1015	Warning	R20-M0-N04-J33		Correctable receiver errors: Node (7,2,1) had 1009216 correctable errors in the X+ direction
ℹ	4/30/08 3:25:52 PM	KERN_1015	Warning	R20-M0-N04-J29		Correctable receiver errors: Node (7,2,0) had 1006701 correctable errors in the X+ direction
ℹ	4/30/08 3:25:52 PM	KERN_1015	Warning	R20-M0-N04-J24		Correctable receiver errors: Node (4,3,1) had 1002667 correctable errors in the X- direction
ℹ	4/30/08 3:25:52 PM	KERN_1015	Warning	R20-M0-N04-J20		Correctable receiver errors: Node (4,3,0) had 1003450 correctable errors in the X- direction
ℹ	4/30/08 3:25:52 PM	KERN_1015	Warning	R20-M0-N04-J25		Correctable receiver errors: Node (4,2,1) had 1001606 correctable errors in the X- direction

Figure 7-6 Display Rows panel

The *Drilldown* tab, shown in Figure 7-7, provides a view of RAS events based on location. Clicking the plus symbol (+) opens (drills down to) the next level of hardware. This view can be helpful when tracking errors that are related to environmental problems.



Filter Options: Most recent 50 events

Location	Fatal	Advisory
R00	2	5
R00-M0	2	4
R00-M0-S	2	4
R00-M0-S-P3	2	2
R00-M0-S-P6	0	2
R00-M1	0	1
R10	6	4
R20	0	21
R30	6	6

Generated: 4/30/08 3:26:31 PM

Figure 7-7 Drilldown view

8

Event Log Analysis

In this chapter, we describe the Event Log Analysis (ELA) tool added in Blue Gene/P V1R4M0. ELA is a system that analyzes Reliability, Availability, and Serviceability (RAS) events and creates alerts to inform the system administrator to take action. ELA includes the following features:

- It is event driven. The analysis is triggered by RAS logging.
- It is modular. Components are defined in Python modules.
- It is flexible. Components are loaded in the ELA pipeline only if they are enabled in the `ela_config.xml` file.
- It is extensible. New plug-in modules can be added.

Important: Configuration and use of ELA requires that Python and PyDB2 are installed on the service node. Appendix F, “Setting up the Event Log Analysis Environment” on page 269 contains instructions for installing those applications.

8.1 Event Log Analysis overview

Figure 8-1 shows that ELA runs on the Blue Gene service node, interacting with the Blue Gene Real-time server and database to fulfill its responsibilities.

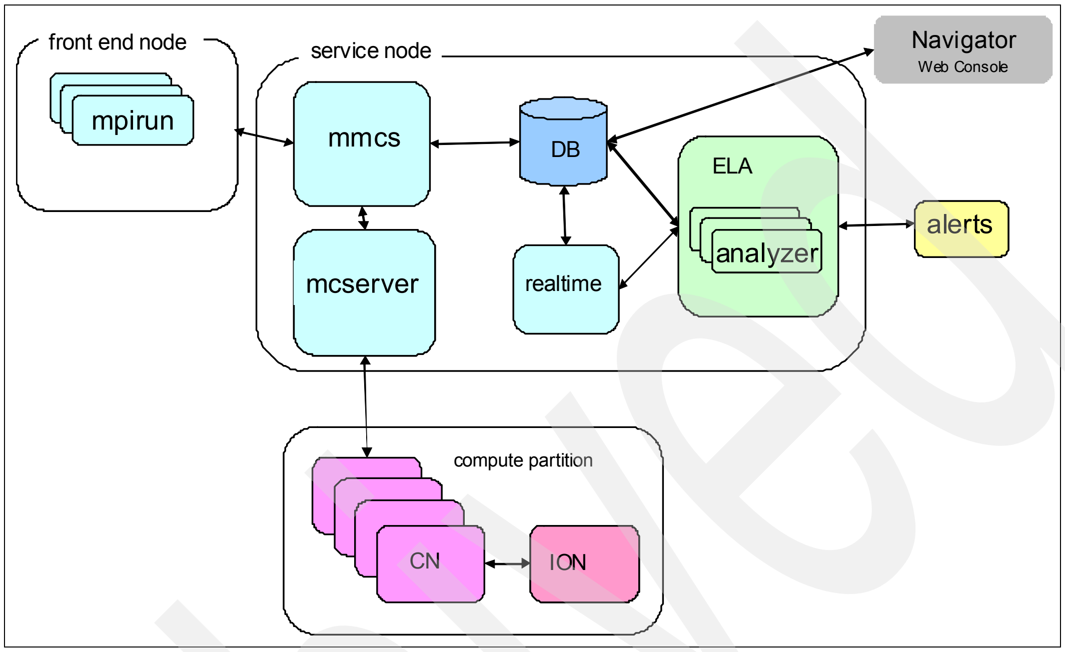


Figure 8-1 ELA overview

Figure 8-2 shows the ELA components and services.

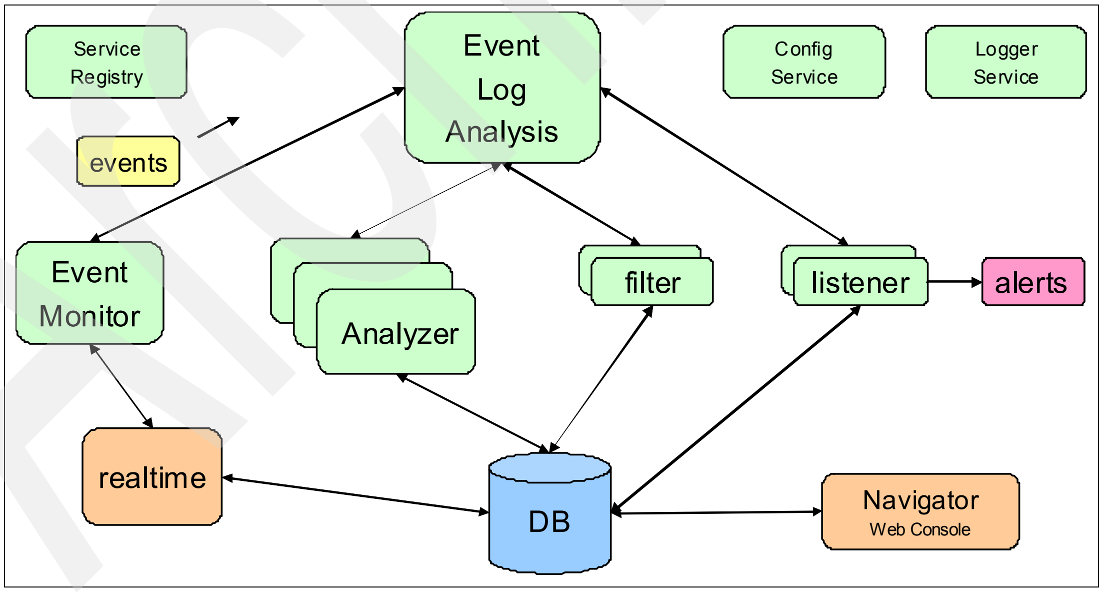


Figure 8-2 ELA components

The following components provide a description of the component's purpose:

- Event monitors, which notify ELA that an RAS event of interest was logged
- Analyzers generate alerts, register for events of interest, and perform analysis when one of those events is logged
- Filters veto an alert if certain conditions exist
- Listeners receive alert notifications and process them

The framework also provides a set of services for the plug-in modules:

- Service registry

  The service registry is used to register and get services. The analyzer registry service is provided by ELA as a means for analyzers to register interest in a set of RAS events. An alert listener service is supplied by ELA as a method for analyzers to post alerts. ELA is responsible for forwarding the alerts to filters. If the alert is not vetoed by a filter, it is forwarded to the listener modules. A queue interface is provided by ELA for event monitors. The event monitors are the producer, putting RAS events on the queue, and ELA is the consumer.

- Configuration service

  The configuration service provides configuration information that is read from various configuration files during ELA initialization.

- Database connection service

  The database connection service provides a shared database connection and cursors. The PyDB2 module is used to get database connections and cursors.

- Logging service

  The logging service is used to return output, either to a log file or standard out.

8.2 Alert and alert history database tables

Alerts are logged in the TBGPALERTLOG table by the DatabaseLogger alert listener, as shown in Example 8-1.

Example 8-1 TBGPALERTLOG schema

```
CREATE TABLE TBGPALERTLOG
(
recid integer GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1),
alerttime timestamp NOT NULL WITH DEFAULT current timestamp,
alertid char(12)  NOT NULL,
severity char(8)  NOT NULL,
location char(64) NOT NULL,
recommendation varchar(256) NOT NULL,
reason varchar(2048) NOT NULL,
source char(64) NOT NULL
);
CREATE ALIAS BGPALERTLOG for TBGPALERTLOG;
```

The TBGPALERTLOG table has the fields:

`recid`	A unique record identifier for the event
`alerttime`	The time at which the alert was logged in the database
`alertid`	Identifier for the alert
`location`	The hardware location
`recommendation`	The action to take
`reason`	Provides more details about the cause of the alert
`source`	The name of the analyzer that created the alert
`severity`	Can be one of the following values:
`fatal`	A fatal error condition was detected
`error`	An error condition was detected
`warning`	Threshold exceeded for non-fatal error
`info`	An informational message (for testing)

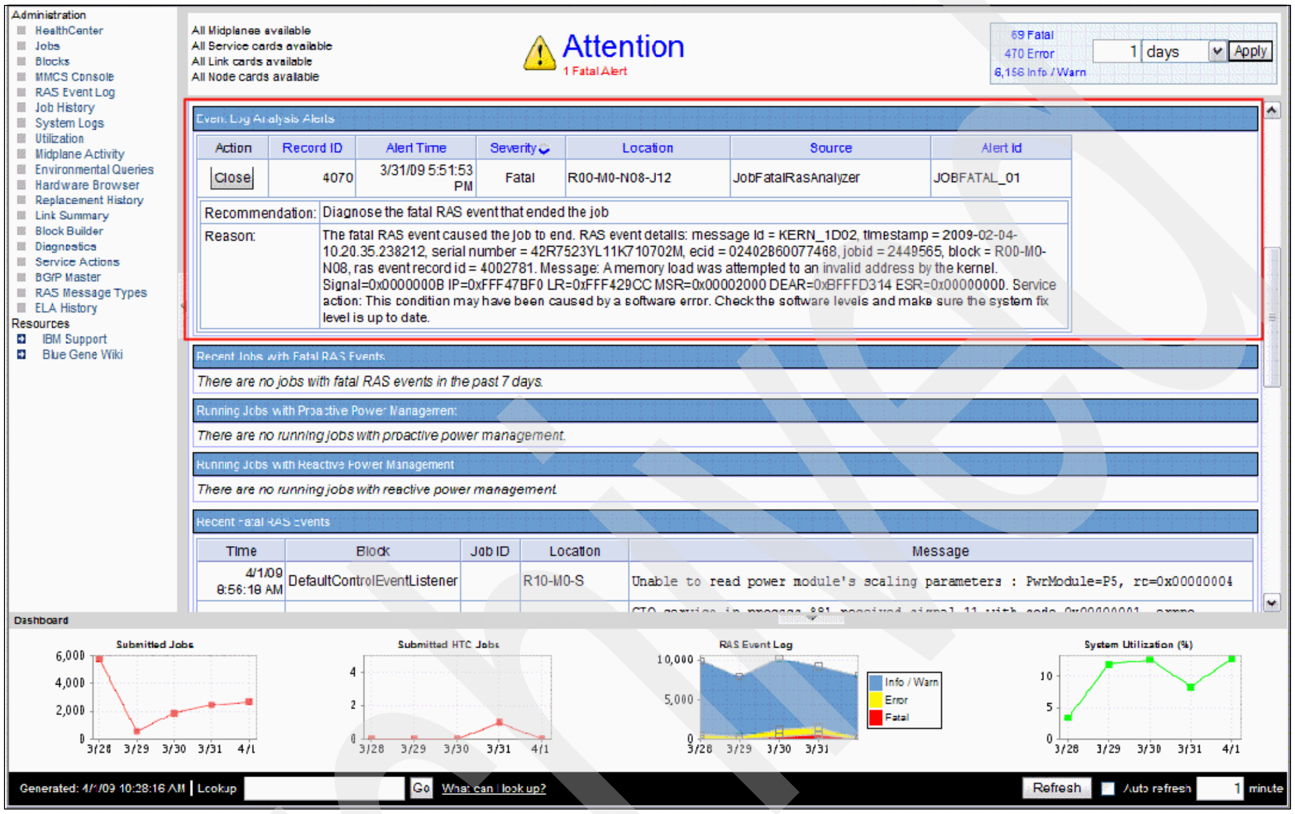
When an alert is deleted from the TBGPALERTLOG table, it is copied to the TBGPALERT_HISTORY table, which is shown in Example 8-2.

Example 8-2 TBGPALERT_HISTORY table

```
CREATE TABLE TBGPALERT_HISTORY
(
entrydate timestamp NOT NULL WITH DEFAULT current timestamp,
recid integer NOT NULL,
alerttime timestamp NOT NULL,
alertid char(12)  NOT NULL,
severity char(8)  NOT NULL,
location char(64) NOT NULL,
recommendation varchar(256) NOT NULL,
reason varchar(2048) NOT NULL,
source char(64) NOT NULL
);
CREATE ALIAS BGPAlert_history for TBGPAlert_history;
```

8.3 Navigator interface for alerts

The Navigator provides a list of ELA alerts in the Health Center (shown in Figure 8-3). When the condition that is associated with the alert is resolved, you can close the alert by clicking the **Close** button in the Action column to move the alert from the TBGPALERTLOG to the TBGPALERT_HISTORY table.



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

Event Log Analysis Alerts

Action	Record ID	Alert Time	Severity	Location	Source	Alert Id
Close	4070	3/31/09 5:51:53 PM	Fatal	R00-M0-N08-J12	JobFatalRasAnalyzer	JOBFATAL_01

Recommendation:	Diagnose the fatal RAS event that ended the job
Reason:	The fatal RAS event caused the job to end. RAS event details: message id = KERN_1D02, timestamp = 2009-02-04-10.20.35.238212, serial number = 42R7523YL11K710702M, ecid = 02402860077468, jobid = 2449565, block = R00-M0-N08, ras event record id = 4002781. Message: A memory load was attempted to an invalid address by the kernel. Signal=0x0000000B IP=0xFFF47BF0 LR=0xFFF429CC MSR=0x00002000 DEAR=0xBFFFD314 ESR=0x00000000. Service action: This condition may have been caused by a software error. Check the software levels and make sure the system fix level is up to date.

Recent Jobs with Fatal RAS Events

There are no jobs with fatal RAS events in the past 7 days.

Running Jobs with Proactive Power Management

There are no running jobs with proactive power management.

Running Jobs with Reactive Power Management

There are no running jobs with reactive power management.

Recent Fatal RAS Events

Time	Block	Job ID	Location	Message
4/1/09 8:56:18 AM	DefaultControlEventListener		R10-M0-S	Unable to read power module's scaling parameters : PwrModule=P5, rc=0x00000004

Generated: 4/1/09 10:28:16 AM

Figure 8-3 Navigator alerts page

8.4 E-mail alerts

The EmailSender event listener sends an e-mail for alerts. You define the e-mail addresses in the `ela_config.xml` file using the syntax shown in Example 8-3.

Example 8-3 Event listener

```
<listener module_name="alert_listener.EmailSender" enabled="true">
      <config_property name="smtp_server" value="localhost"/>
      <email_from  name="ELA Server" userid="ela_server@abc.com"/>
      <email_to name="SysAdmin1" userid="admin1@abc.com"/>
      <email_to name="SysAdmin2" userid="admin2@abc.com"/>
   </listener>
```

The e-mail subject line includes a header BG ELA Alert, alert ID, hardware location, and recommendation. The e-mail body includes the alert details. Example 8-4 provides a sample of an e-mail notification.

Example 8-4 Sample e-mail

```
Subject: BG ELA Alert: THRESH_01 R30-M0-S Schedule part replacement

Severity: warning
Location: R30-M0-S
Recommendation: Schedule part replacement
Reason: Error threshold of 1 exceeded.  RAS event details: message id = CARD_0414,
timestamp = 2009-02-04-11.34.10.483952, serial number = 42R8299YL10K7152007, ecid
= None, jobid = None, block = DefaultControlEventListener, ras event record id =
4002898.  Service action: Replacement of the part at this location is recommended
if 1 or more errors have occurred.
Source: ThresholdExceededAnalyzer
```

8.5 Extending ELA

ELA modules are written in the Python scripting language. Python was selected because of its popularity with system administrators, the number of built-in libraries that it contains, and the availability of external database libraries such as PyDB2.

ELA provides a plug-in framework with four types of plug-in modules:

- Alert listeners
- Alert filters
- Event analyzers
- Event monitors

The modules are arranged in a pipeline that is managed by the ELA program, as shown in Figure 8-4.

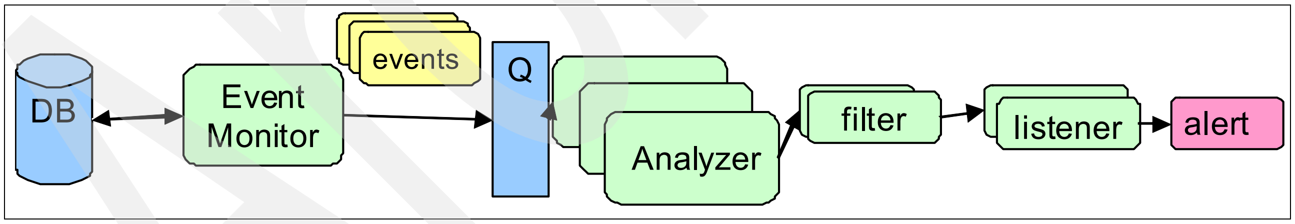


Figure 8-4 ELA framework

The four module types are placed in their respective subdirectories as a design practice:

- `/bgsys/drivers/ppcfloor/tools/ela/alert_listener`
- `/bgsys/drivers/ppcfloor/tools/ela/alert_filter`
- `/bgsys/drivers/ppcfloor/tools/ela/analyzer`
- `/bgsys/drivers/ppcfloor/tools/ela/event_monitor`

The `ela_config.xml` file defines the modules and whether or not they are enabled. The enabled modules are instantiated when ELA is initialized.

8.5.1 Alert listeners

Alert listeners process alerts. A DatabaseLogger, EmailSender, and SampleListener are provided. You can develop new listeners to adapt the alert notification to your IT infrastructure. For example, you can develop a listener that posts alerts to a Twitter account and have your administrators follow that account on their mobile devices. You define alert listeners in the `/bgsys/drivers/ppcfloor/tools/ela/alert_listener` directory.

The AlertListener base class is defined in the `AlertListener.py` module, as shown in Example 8-5

Example 8-5 AlertListener base class

```
class AlertListener:
    '''The AlertListener class listens for alerts.
    '''
    def alert(self, alert):
        '''To send an alert to a listener.'''
        Return
```

A sample listener implementation, shown in Example 8-6, is provided in the SampleListener module. The sample listener logs the event in a text log.

Example 8-6 Sample listener implementation

```
class SampleListener(AlertListener):
    '''The SampleListener is a sample for alert listeners.  It receives
    an alert and logs it.
    '''
    def __init__(self):
        '''The constructor.
        '''
        config_service = ServiceRegistry.getService('config_service')
        self.verbose = config_service.get('alert_listener.SampleListener.verbose')
        return

    def alert(self, alert):
        '''Log (send) an alert.
        '''
        # log alert details
        logger = ServiceRegistry.getService('log_service')
        logger.info('Alert ID: ' + alert.alertid + ' Location: ' + alert.location
+ ' Recommendation: ' + alert.recommendation + ' Severity: ' + alert.severity)
        if self.verbose == 'true':
            logger.info('\tReason: ' + alert.reason)
        return
```

8.5.2 Alert filters

Alert filters veto alerts if certain conditions exist. An ActiveAlertCeilingFilter, CommonModeFilter, and DuplicateFilter are provided. You define alert filters in the `/bgsys/drivers/ppcfloor/tools/ela/alert_filter` directory. The AlertFilter base class, as shown in Example 8-7, is defined in the `AlertFilter.py` module.

Example 8-7 AlertFilter base class

```
class AlertFilter:
    '''The AlertFilter class filters alerts.
    '''

    def filter(self, alert):
        '''To filter (veto) an alert return True, else return False'''
        return
```

The implementation of the duplicate filter, as shown in Example 8-8, vetoes sending an alert if another alert for the same hardware location is already logged.

Example 8-8 DuplicateFilter class

```
class DuplicateFilter(AlertFilter):
    '''The DuplicateFilter will filter any duplicate alerts for the same location.
    '''

    def __init__(self):
        ''' The constructor.
        '''
       self.duplicate_query = "select location from tbgpalertlog where location =
?"
        return

    def filter(self, alert):
       ''' Indicate whether or not an alert for the same location exist (has been
logged).
        Return true if it exists, otherwise return false.
        '''
        cursor = ServiceRegistry.getService('dbconn').cursor()
        cursor.execute(self.duplicate_query, alert.location)
        row = cursor.fetchone()
        if (row == None):
            return False
        else:
            logger = ServiceRegistry.getService('log_service')
            logger.info('filtered alert ID: ' + alert.alertid + ' location: ' +
alert.location)
            return True
```

8.5.3 RAS event analyzers

These analyzers analyze RAS events and send an alert if necessary. Multiple analyzer implementations are provided. You define analyzers in the `/bgsys/drivers/ppcfloor/tools/ela/analyzer` directory. The Analyzer base class is defined in the `Analyzer.py` module, as shown in Example 8-9.

Example 8-9 Analyzer base class

```
class Analyzer:

    '''The Analyzer class is the base class for modules that
    analyze events to determine if an alert should be raised.
    '''
    def __init__(self):
        '''The constructor.'''
        return

    def getName(self):
        '''Get the name of the analyzer.'''
        return

    def getAlertIDs(self):
        '''Return a dictionary where the keys are the list of possible
        alert IDs produced by this analyzer and the values are the
        severity of the alert.'''
        return

    def getMsgIDs(self):
        '''Get the list of RAS event message ids of interest
        to this analyzer.'''
        return

    def analyze(self, event):
        '''The analyze method performs the analysis and determines if alerts
        should be created for conditions that require administrator action.'''
        return
```

Example 8-10 shows a sample analyzer.

Example 8-10 Sample analyzer

```
class SampleAnalyzer(BaseAnalyzer):

    '''The SampleAnalyzer class registers for all ras events and
    sends alerts for them as they arrive.
    '''
    def __init__(self):
        '''The constructor.
        '''
        self._name = 'SampleAnalyzer'
        self.alertIDsev = {'SAMPLE_01':'info'}
        self.recommendation = 'Sample alert - ignore'

        # Register to receive all ras events (note that a typical analyzer
        # would be interested in a small subset of the events).
        rows = self.executeQuery("select msg_id from tbgperrcodes")
        self.msgIDs = list()
        for r in rows:
            self.msgIDs.append(r[0].strip())
        ServiceRegistry.getService('analyzer_registry').registerAnalyzer(self,
self.msgIDs)
        return

    def getName(self):
        '''Return name of the analyzer.
        '''
        return self._name

    def getAlertIDs(self):
        '''Return list of alert IDs for the analyzer.
        '''
        return self.alertIDsev

    def getMsgIDs(self):
        '''Return list of message IDs for the analyzer.
        '''
        return self.msgIDs

    def analyze(self, event):
        '''Analyze a RAS event.
        '''
        # get the ras event details
        cursor = ServiceRegistry.getService('dbconn').cursor()
        location, severity, serialnumber, ecid, event_time, jobid, block, msgText
= self.rasDetailQuery(cursor, int(event.recid))

        # analyze the condition and send an alert if necessary
        # Only send an alert if it has a location
        if location != None:
            self.sendAlert('SAMPLE_01', self.alertIDsev['SAMPLE_01'], location,
self.recommendation, ("This alert was generated as part of a ELA test and can be
ignored.  \nRAS event details: message id = %s, severity = %s, timestamp = %s,
serial number = %s, ecid = %s, jobid = %s, block = %s, ras event record id = %s.
\nMessage: %s." % (event.msgid, severity, event_time, serialnumber, ecid, jobid,
block, event.recid, msgText) ), self._name)
        return
```

8.5.4 RAS event monitors

RAS event monitors notify ELA when a RAS event of interest is logged. You can enable only one event monitor. The RealtimeEventMonitor registers for RAS events using the Blue Gene Real-time API. A PeriodicRASEventMonitor is provided for cases when Real-time is not running. A HistoricRASEventQuery is provided to analyze the set of RAS events that are logged during a start and stop time and date. This monitor can help verify that an analyzer is handling a set of events correctly.

You define event monitors in the `/bgsys/drivers/ppcfloor/tools/ela/event_monitor` directory. The EventMonitor base class is defined in `EventMonitor.py`, as shown in Example 8-11.

Example 8-11 EventMonitor base class

```
class EventMonitor:
    '''The EventMonitor class defines the interface for the
    event monitor.
    '''
    def start(self, msgidList):
        ''' Start the event monitor.
        '''
        return

    def stop(self):
        ''' Stop the event monitor.
        '''
        return
```

9

Diagnostics

This chapter describes the diagnostics suite that is provided with Blue Gene/P. We demonstrate how to run the Blue Gene/P Diagnostic suite from the Blue Gene Navigator interface and from the command line.

9.1 System diagnostics

The Blue Gene/P Diagnostic suite interacts with the MMCS server to run tests on the Blue Gene/P hardware. The Diagnostic test suite provides a method of locating hardware failures. These tests aid in categorizing and quantifying hardware failures and determining whether the failures are either an anomaly or systematic.

Diagnostics must be executed from `/bgsys/driver/ppcfloor/bin` on the service node or from Blue Gene Navigator. The Navigator is an administrative Web application that you can use to configure and execute Diagnostic runs, view run status, and browse Diagnostic run history. For more information, see Chapter 2, "Blue Gene Navigator" on page 17.

Diagnostics include tests for:

- Memory subsystem
- Compute logic
- Instruction units
- Floating point units
- Torus and collective network
- Internal and external chip communication
- Global interrupts

9.1.1 Requirements

The service node must be running SUSE® Linux Enterprise Server 10. To control the rack, the MMCS server and mcServer must be running. Your TCP/IP network must be functional because the Diagnostics suite uses TCP/IP sockets to communicate between the service node and the system.

The Diagnostics suite installs automatically as a part of the RPM installation of the system. The default directory for the Diagnostics suite code is `/bgsys/drivers/ppcfloor/diags`. The code is updated with fix packs as well as Blue Gene/P releases and modifications.

Users who execute diagnostics must have read and execute authority to the `/bgsys/drivers/ppcfloor/diags` directory and the subdirectories that contain the test cases. The output that the diagnostics generate is written to the `/bgsys/logs/BGP/diags` directory. Each time you run the diagnostics, a new directory is created in the `/bgsys/logs/BGP/diags` directory. These new directories use the naming convention *yymmdd_hhmmss*`_identifier`. Inside the new directories, there is a file named `diags.log` that contains output from the entire run. In addition, there is a subdirectory for each test case that was included in the diagnostic run. The subdirectories use the naming convention *TestcaseName_BlockName_timestamp*. Any time a test case runs, the `output.log` and `rasFile.log` files are written to the test case's subdirectory. If the test case completes successfully, these logs are deleted to save disk space. The user that initiates the Diagnostic run must have all authority to each of these directories.

9.2 Diagnostic test cases

Diagnostics test cases are designed to test all aspects of the hardware. You might have occasions when there is not enough time to run all of the tests. You might also have an issue that requires that you run a specific type of test, only on certain hardware. To accommodate these instances there are *test buckets*. These test buckets are groups of test cases based on time or a specific type of hardware. When running the Diagnostics suite from the Navigator, you can base the run on the amount of time that you have allowed.

> **Note:** Run these tests with care. For example, the BPC BIST test (bpclbist) is very intensive when it comes to service node resources and can prevent the control system from booting blocks in a timely manner. Tweaking the `boot-node-timeout-multiplier` setting can alleviate this effect to some degree. However, it is recommended that for best results you do not run the BPC BIST test when attempting to boot other blocks.
>
> The BPL BIST test (`bpllbist`) renders link chips unusable by other blocks for any purposes while the diagnostic is running. For example, attempting to use link chips running LBIST for pass through will not work. If you are running BPL BIST, be sure that other blocks are kept from using the affected link cards. The BPL BIST test is not included in any of the buckets to avoid causing problems for link cards that are used for pass through. Running diagnostics from the command line allows more options.

The following test buckets are available on the command line:

`checkup`	A series of tests that are designed to provide maximum coverage for approximately 30 minutes of runtime
`small`	A quick diagnostics run that provides low fault detection
`medium`	A medium length diagnostics run that provides fault detection
`large`	A long length diagnostics run that provides high fault detection
`complete`	Runs all diagnostics, takes the longest time and provides the highest fault detection
`servicecard`	All service card-specific tests
`nodecard`	All node card-specific tests
`linkcard`	All link card-specific tests
`memory`	Tests that target the memory subsystem on the nodes.
`processor`	Tests that target the node processors
`torus`	Tests that target the torus network
`collective`	Tests that target the collective network
`barriers`	Tests that target the barrier hardware
`regulators`	Tests that target the DC-DC power regulators on the node cards
`Ethernet`	Tests that target the Ethernet hardware

Table 9-1 provides a list of the current test cases that are available, the average amount of run time in minutes, a brief description, and the buckets in which they run.

Table 9-1 Test cases

Name	Run time	Description	Bucket
svccardenv	5	Checks voltages, currents, temperatures, clocks, and other environmental parameters on the service card.	small, medium, large, complete, servicecard
readncpower	1	Performs a quick check of the node card power modules.	small, medium, large, complete, nodecard
nodecardenv	5	Checks voltages, currents, temperatures, clocks, and other environmental parameters on the node card.	small, medium, large, complete, nodecard
readlcpower	1	Performs a quick check of the link card power modules.	small, medium, large, complete, linkcard
linkcardenv	5	Checks voltages, currents, temperatures, clocks, and other environmental parameters on the link card.	small, medium, large, complete, linkcard
bpllbist	2	Runs LBIST (Logical Built-In Self-Test) on the BPL ASICs. **Note**: Do not run this test on link cards performing pass through.	
bpclbist	6	Runs LBIST (Logical Built-In Self-Test) on the BPC ASICs.	small, medium, large, complete, memory, processor, torus, collective, barriers
ddr_bitfail	4	Tests the PPC450 by writing to and reading from all DDR memory locations, flagging all failures (data, ECC, chip select, address). The ddr_bitfail test does a simple memory test and prints a log description that attempts to identify the failing component. It identifies specific failing bits by ASIC and DRAM pin when possible.	small, medium, large, complete, checkup, memory
ms_gen_short	8	Tests the cache hierarchy within the nodes. This is a more complete memory test that checks both BPC ASIC function and the external DRAM. Error messages in the log attempt to report what failed but are not always successful.	small, medium, large, complete, checkup, memory
conch	10	Checks memory consistency.	small, medium, large, complete, memory, processor
dgemm_l3	10	Tests the on-chip functionality of the floating point unit on the BPC ASIC.	medium, large, complete, memory
dgemm_dram	12	Tests the BPC ASIC, FPU, and memory subsystem on the compute nodes.	large, complete, nodecard, memory
dgemm_pulse	14	Tests the BPC ASIC, FPU, and memory subsystem on the compute nodes, but it does so through bursts of activity across the nodes, exercising the power modules.	medium, large, complete, nodecard, checkup, memory, barriers, regulators
gi_basic	3	Tests the global interrupt network.	large, complete, barriers

Name	Run time	Description	Bucket
gi_comp	5	Tests the global interrupt network.	medium, large, complete, barriers
gi_node	5	Tests the global interrupt network.	medium, large, complete, barriers
gi_link	5	Tests the global interrupt network.	large, complete, barriers
trash	6	Runs random instruction streams to exercise the PPC cores.	large, complete, processor
tnk	16	Exercises both logic and memory paths on the compute nodes from a collection of several tests.	medium, large, complete, processor
tr_diag_loopback	5	Tests the following major components with a single node test of the tree unit: ▶ SRAM and its ECC correction ▶ ALU functionality within the tree arbiter ▶ General packet routing Any errors are local to the ASIC.	medium, large, complete, collective
ts_dma_loopback	5	Tests the DMA loopback.	medium, large, complete, checkup, torus
tr_and_dma_ connectivity	8	Tests the tree and DMA connectivity.	medium, large, complete, checkup, torus, collective, barriers
XEMAC_loopback_ polling	5	Transmits a single TCP packet, polls the Received Frames register for successful reception of the packet, and verifies the content of the received packet. Configured to be in internal loopback mode. Both the transmission and reception are done by a single core.	medium, large, complete, Ethernet
XEMAC_loopback_ interrupt	5	Transmits a single TCP packet, receives an interrupt that a packet has been received, and verifies the content of the received packet. Configured to be in internal loopback mode. The transmission is done by core 0 and the reception is done by core 1.	medium, large, complete, Ethernet
XEMAC_loopback_ multi_core	5	Transmits a stream of packets by one core and receives the packets by another core. Configured to be in internal loopback mode. The Received Frames register is polled for successful reception of the received packets.	medium, large, complete, Ethernet
PHY_loopback	5	Transmits a stream of packets by one core and receives the packets by another core. Configured to be in loop back mode. The Received Frames register is polled for successful reception of the received packets. Tests the entire path, as follows: TOMAL0 ∅ XEMAC0 ∅ XGXSPCS0 ∅ PHY0 ∅ XGXSPCS0 ∅ XEMAC0 ∅ TOMAL0.	medium, large, complete, Ethernet
boot	4	Performs a boot with the default images. Any errors are analyzed by diagnostics.	complete, Ethernet
neighbor	4	Sends each node a message to its six neighbors sequentially. A basic torus test. Hangs in the test indicate torus communication problems, which is how the test determines bad torus links.	checkup, complete, nodecard, multinode

9.3 Running diagnostics

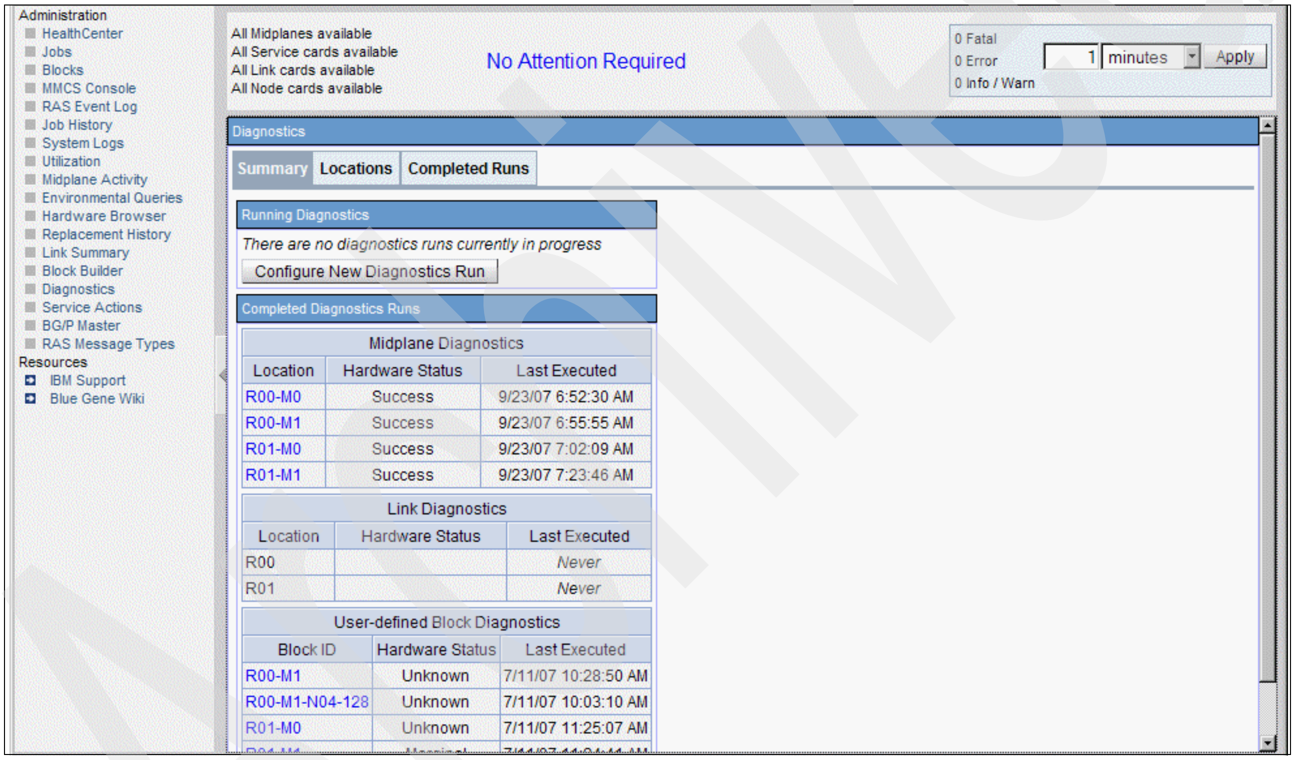
As mentioned previously, there are two methods to initiate diagnostics:

- Using the Blue Gene Navigator Diagnostics interface
- Starting the diagnostics from a command line on the service node

To run diagnostics, you must have the authorities listed in 9.1, “System diagnostics” on page 168.

9.3.1 Using the Navigator Diagnostics interface

Clicking **Diagnostics** in the navigation pane of the Navigator opens the page shown in Figure 9-1. The most recent runs for midplane, link, and user-defined block diagnostics are listed on this page. Each Location or Block ID is a link that takes you to the page that contains the history of all Diagnostic runs for that specific location or Block ID.



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

No Attention Required

Running Diagnostics

There are no diagnostics runs currently in progress

Completed Diagnostics Runs

Midplane Diagnostics

Location	Hardware Status	Last Executed
R00-M0	Success	9/23/07 6:52:30 AM
R00-M1	Success	9/23/07 6:55:55 AM
R01-M0	Success	9/23/07 7:02:09 AM
R01-M1	Success	9/23/07 7:23:46 AM

Link Diagnostics

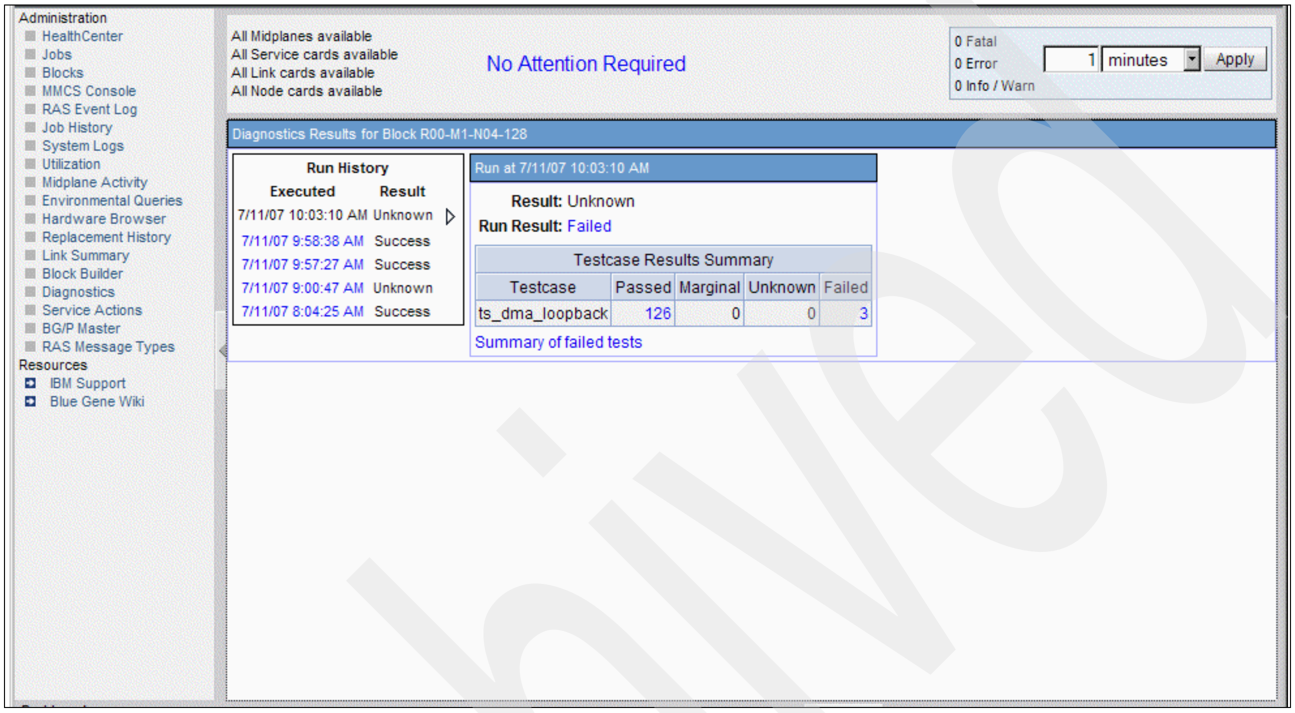
Location	Hardware Status	Last Executed
R00		*Never*
R01		*Never*

User-defined Block Diagnostics

Block ID	Hardware Status	Last Executed
R00-M1	Unknown	7/11/07 10:28:50 AM
R00-M1-N04-128	Unknown	7/11/07 10:03:10 AM
R01-M0	Unknown	7/11/07 11:25:07 AM

Figure 9-1 Diagnostics home page

Each Location or Block ID is a link that takes you to the page that contains the history of all Diagnostic runs for that specific location or Block ID. For this example, we selected the R00-M1-N04-128 user-defined block link, which opens the page shown in Figure 9-2. By clicking the date and time stamp link on the left, you can view the result for each test case that was included in the run. In our example, the run included only one test case, `ts_dma_loopback`.



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

No Attention Required

0 Fatal
0 Error
0 Info / Warn

1 minutes

Diagnostics Results for Block R00-M1-N04-128

Run History

Executed	Result
7/11/07 10:03:10 AM	Unknown
7/11/07 9:58:38 AM	Success
7/11/07 9:57:27 AM	Success
7/11/07 9:00:47 AM	Unknown
7/11/07 8:04:25 AM	Success

Run at 7/11/07 10:03:10 AM

Result: Unknown
Run Result: Failed

Testcase Results Summary

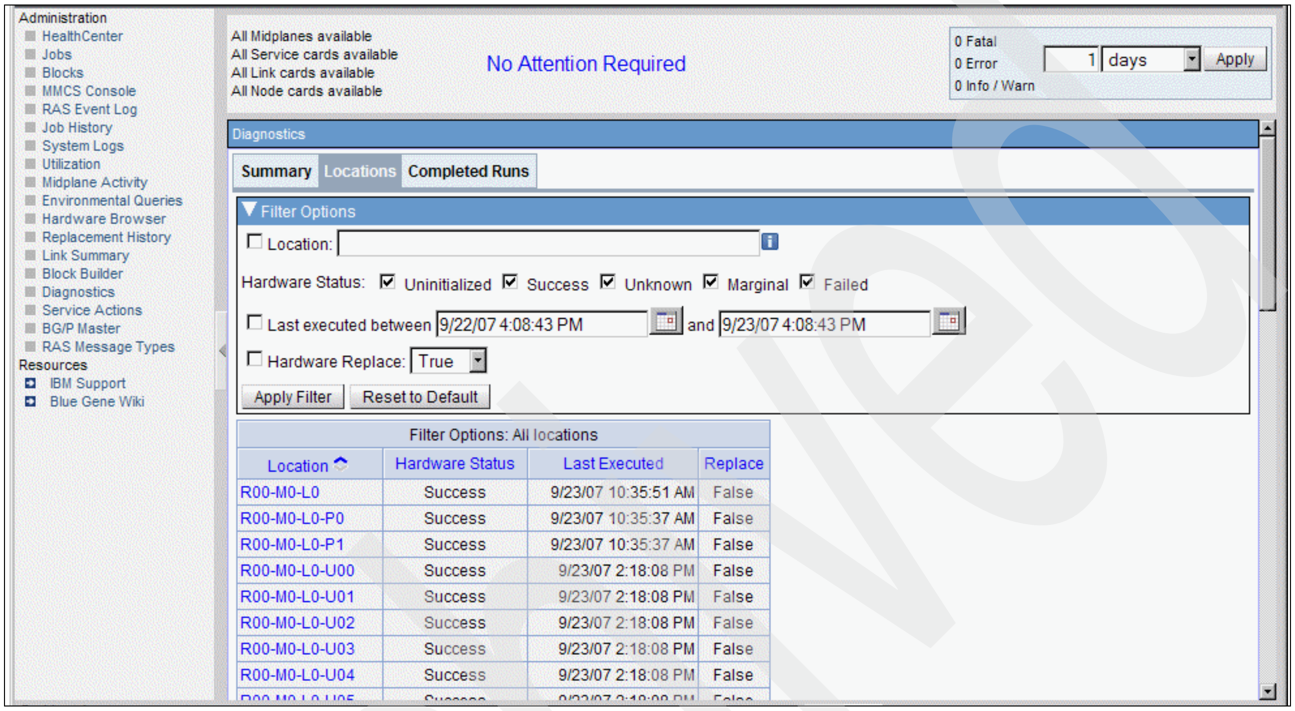
Testcase	Passed	Marginal	Unknown	Failed
ts_dma_loopback	126	0	0	3

Summary of failed tests

Figure 9-2 User-defined block page

The Diagnostics main page includes three tabs across the top of the main content area:

- The Summary tab is the page that you see when first clicking the **Diagnostics** link.
- The Locations tab, shown in Figure 9-3, provides a view of all the hardware on which diagnostics has run. Each location string on the left is a link to the summary of diagnostics that have run on that location. You can filter the results on, and sort by, each of the column heads.



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

No Attention Required

0 Fatal
0 Error
0 Info / Warn

1 days

Location:

Hardware Status: ☑ Uninitialized ☑ Success ☑ Unknown ☑ Marginal ☑ Failed

☐ Last executed between 9/22/07 4:08:43 PM and 9/23/07 4:08:43 PM

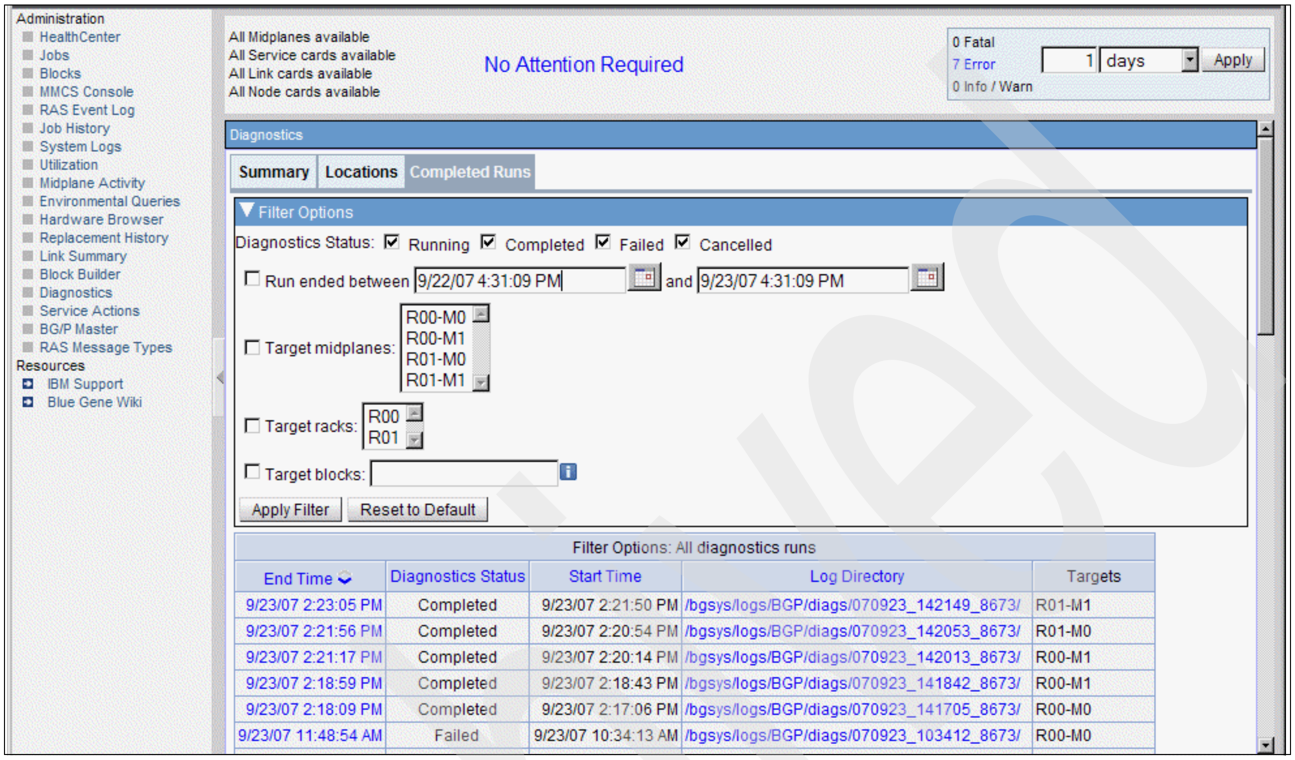
☐ Hardware Replace: True

Filter Options: All locations

Location	Hardware Status	Last Executed	Replace
R00-M0-L0	Success	9/23/07 10:35:51 AM	False
R00-M0-L0-P0	Success	9/23/07 10:35:37 AM	False
R00-M0-L0-P1	Success	9/23/07 10:35:37 AM	False
R00-M0-L0-U00	Success	9/23/07 2:18:08 PM	False
R00-M0-L0-U01	Success	9/23/07 2:18:08 PM	False
R00-M0-L0-U02	Success	9/23/07 2:18:08 PM	False
R00-M0-L0-U03	Success	9/23/07 2:18:08 PM	False
R00-M0-L0-U04	Success	9/23/07 2:18:08 PM	False
R00-M0-L0-U05	Success	9/23/07 2:18:08 PM	False

Figure 9-3 Diagnostics Location tab

- ► The Completed Runs tab, Figure 9-4, shows a history of all the runs of diagnostics that have run to completion. Included on the page are Filter Options that allow you to trim the results that are shown on the page according to the status, time of run, and the target of the run (midplane, rack, or block).



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

No Attention Required

0 Fatal
7 Error
0 Info / Warn

1 days

Diagnostics Status: ☑ Running ☑ Completed ☑ Failed ☑ Cancelled

☐ Run ended between 9/22/07 4:31:09 PM and 9/23/07 4:31:09 PM

☐ Target midplanes: R00-M0 R00-M1 R01-M0 R01-M1

☐ Target racks: R00 R01

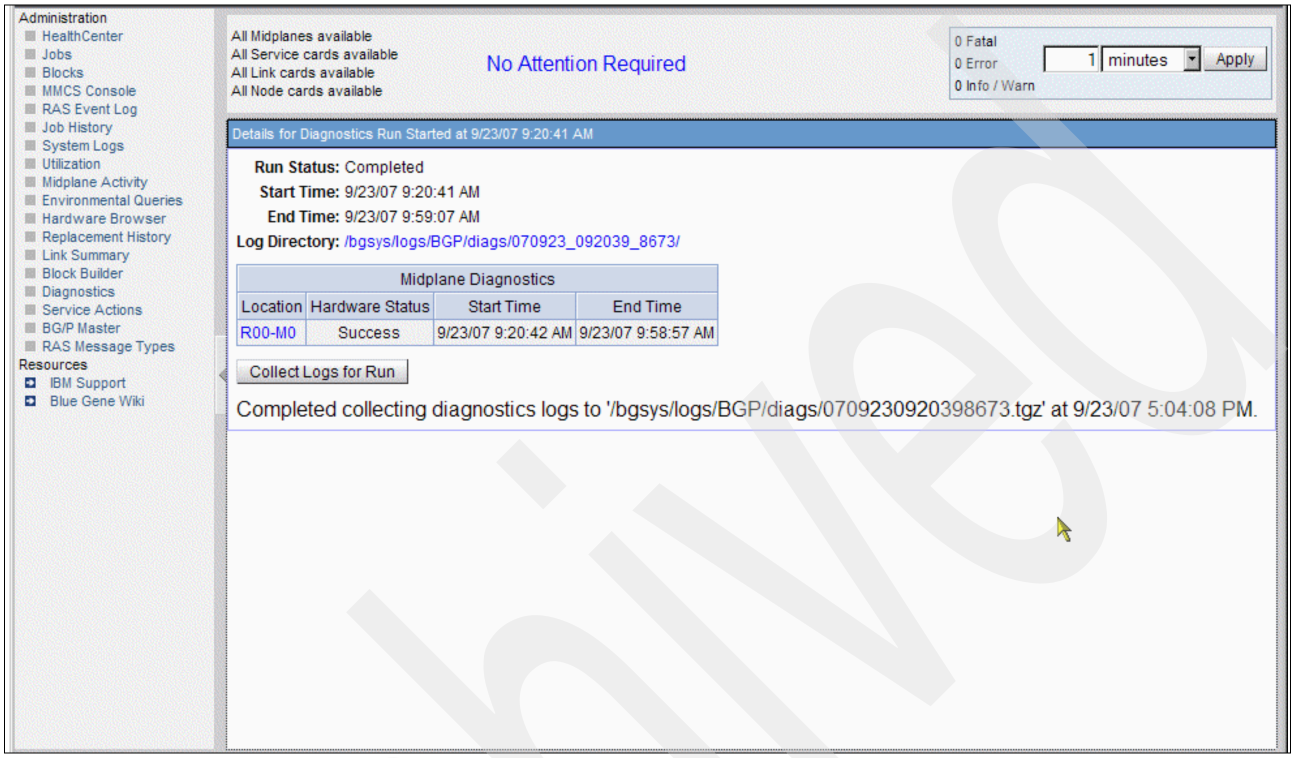
☐ Target blocks:

Filter Options: All diagnostics runs

End Time	Diagnostics Status	Start Time	Log Directory	Targets
9/23/07 2:23:05 PM	Completed	9/23/07 2:21:50 PM	/bgsys/logs/BGP/diags/070923_142149_8673/	R01-M1
9/23/07 2:21:56 PM	Completed	9/23/07 2:20:54 PM	/bgsys/logs/BGP/diags/070923_142053_8673/	R01-M0
9/23/07 2:21:17 PM	Completed	9/23/07 2:20:14 PM	/bgsys/logs/BGP/diags/070923_142013_8673/	R00-M1
9/23/07 2:18:59 PM	Completed	9/23/07 2:18:43 PM	/bgsys/logs/BGP/diags/070923_141842_8673/	R00-M1
9/23/07 2:18:09 PM	Completed	9/23/07 2:17:06 PM	/bgsys/logs/BGP/diags/070923_141705_8673/	R00-M0
9/23/07 11:48:54 AM	Failed	9/23/07 10:34:13 AM	/bgsys/logs/BGP/diags/070923_103412_8673/	R00-M0

Figure 9-4 Completed runs tab

To get more information about a specific run, you can click the link in the Log Directory column, which opens the log files for that run. To see the details for a specific run, click the link in the End Time column. See Figure 9-5. Click **Collect Logs for Run** at the bottom of the page to generate a `.tar` file with the logs from the run. The tar file is generated in the `/bgsys/logs/BGP/diags` directory.



Details for Diagnostics Run Started at 9/23/07 9:20:41 AM

Run Status: Completed
Start Time: 9/23/07 9:20:41 AM
End Time: 9/23/07 9:59:07 AM
Log Directory: /bgsys/logs/BGP/diags/070923_092039_8673/

Midplane Diagnostics			
Location	Hardware Status	Start Time	End Time
R00-M0	Success	9/23/07 9:20:42 AM	9/23/07 9:58:57 AM

Collect Logs for Run

Completed collecting diagnostics logs to '/bgsys/logs/BGP/diags/0709230920398673.tgz' at 9/23/07 5:04:08 PM.

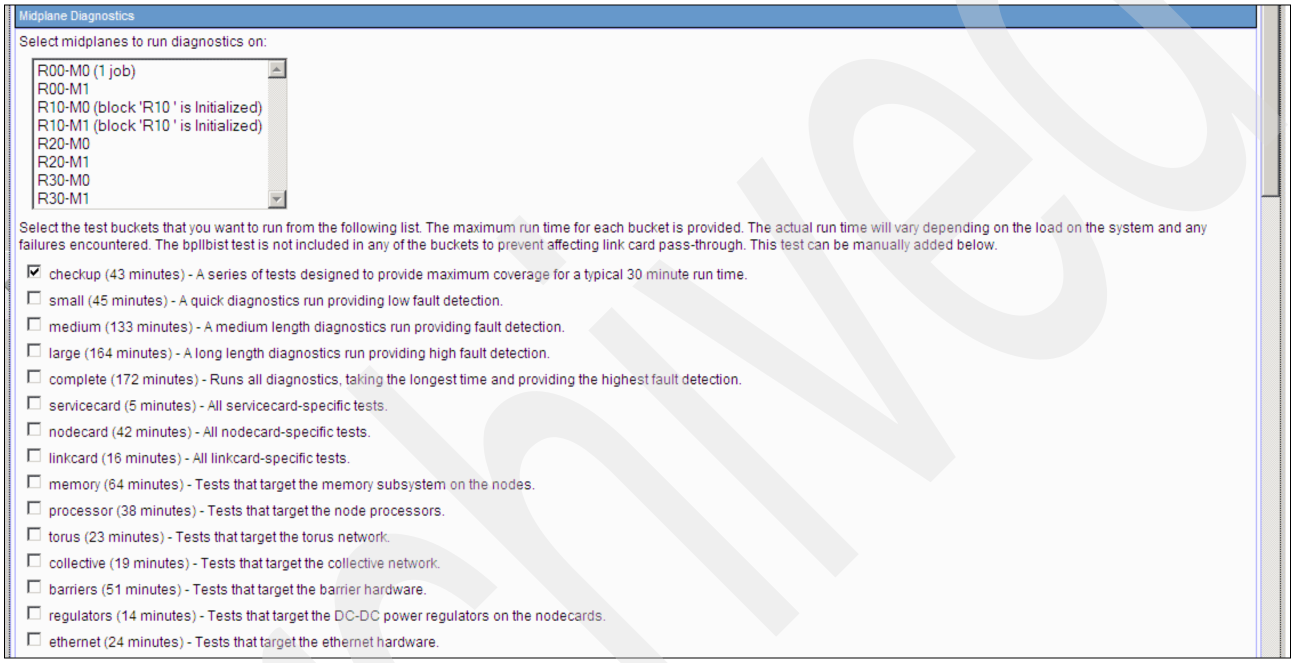
Figure 9-5 Diagnostics run details

Starting a Diagnostic run

Restriction: The diagnostics that you submit from the Navigator share a common pool of available memory. As such, if you test a large set of hardware concurrently, out of memory errors can occur. If more memory is required, then you need to submit diagnostics runs from the command line. When submitted from the command line, each run of diagnostics is allocated a separate pool of memory.

To start a diagnostics run from the Blue Gene Navigator, follow these steps:

1. Go to the Summary tab. Under the Running Diagnostics heading, click **Configure New Diagnostics Run** to open the page shown in Figure 9-6.



Midplane Diagnostics

Select midplanes to run diagnostics on:

R00-M0 (1 job)
R00-M1
R10-M0 (block 'R10 ' is Initialized)
R10-M1 (block 'R10 ' is Initialized)
R20-M0
R20-M1
R30-M0
R30-M1

Select the test buckets that you want to run from the following list. The maximum run time for each bucket is provided. The actual run time will vary depending on the load on the system and any failures encountered. The bpllbist test is not included in any of the buckets to prevent affecting link card pass-through. This test can be manually added below.

- [x] checkup (43 minutes) - A series of tests designed to provide maximum coverage for a typical 30 minute run time.
- [ ] small (45 minutes) - A quick diagnostics run providing low fault detection.
- [ ] medium (133 minutes) - A medium length diagnostics run providing fault detection.
- [ ] large (164 minutes) - A long length diagnostics run providing high fault detection.
- [ ] complete (172 minutes) - Runs all diagnostics, taking the longest time and providing the highest fault detection.
- [ ] servicecard (5 minutes) - All servicecard-specific tests.
- [ ] nodecard (42 minutes) - All nodecard-specific tests.
- [ ] linkcard (16 minutes) - All linkcard-specific tests.
- [ ] memory (64 minutes) - Tests that target the memory subsystem on the nodes.
- [ ] processor (38 minutes) - Tests that target the node processors.
- [ ] torus (23 minutes) - Tests that target the torus network.
- [ ] collective (19 minutes) - Tests that target the collective network.
- [ ] barriers (51 minutes) - Tests that target the barrier hardware.
- [ ] regulators (14 minutes) - Tests that target the DC-DC power regulators on the nodecards.
- [ ] ethernet (24 minutes) - Tests that target the ethernet hardware.

Figure 9-6 Configure Diagnostic run

2. Select the midplanes that you want to include in the run. You can select multiple midplanes by holding the Ctrl key and selecting each of the midplanes on which you want to run diagnostics.
3. Next, select the test bucket that you want to run. You can select multiple test buckets.
4. Add or remove test cases individually by toggling the check mark on or off in the box on the left-hand side of the page.
5. Select the Run Options at the bottom of the page, as shown in Figure 9-7:

   By default, the system configures the run to use all of the I/O nodes on the system. If you have a system that does not have all of the I/O nodes plugged in to the Ethernet, change the “Pset ratio override” option to reflect the I/O node to compute node ratio of your system. To override the system, select this option, and then select the ratio, from the drop-down menu, that you want to use.

   If you want the diagnostics run to stop on the first error that it encounters, select the second option.

   Selecting the “Save all the output from diagnostics” option forces diagnostics to save all the output from the run.

By default, diagnostics allow only two midplanes to run in parallel per processor in the service node. If you want more, select the “Midplanes per service node processor” option, and enter the desired number of concurrent midplane runs in the text box.

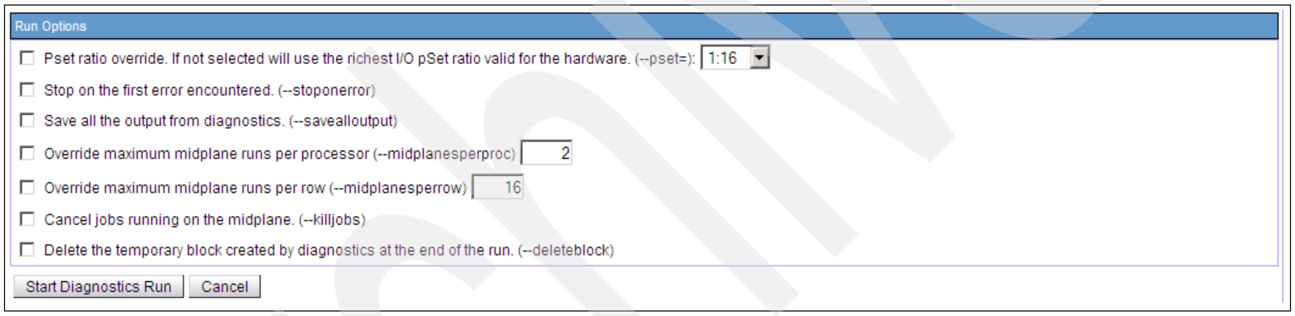
By default, diagnostics allow only 16 midplanes to run in parallel within a row of the system. If you want to enter a different value, select the “Midplanes per row” option, and enter the number of concurrent midplane runs in the text box.

Beginning in Blue Gene/P V1R4M0, you can direct diagnostics to cancel jobs (`--killjobs`) and free the blocks that are running on a midplane that might prevent the diagnostics run from initializing. This option is valid only when running diagnostics on a full midplane.

Note: If you attempt to select the `--killjobs` option on a diagnostics run that is smaller than a midplane, the run is aborted.

Also introduced in Blue Gene/P V1R4M0, you can delete the blocks that diagnostics creates for midplane runs. If you do not delete the block (by selecting the “Delete the temporary block created by diagnostics at the end of the run” option), the block remains available for later use.

6. To start the run, click **Start Diagnostics Run** or click **Cancel** to abort the run. A message prompts you to verify that you want to start the Diagnostic Run. Click **OK** to continue.



☐ Pset ratio override. If not selected will use the richest I/O pSet ratio valid for the hardware. (--pset=): 1:16

☐ Stop on the first error encountered. (--stoponerror)

☐ Save all the output from diagnostics. (--savealloutput)

☐ Override maximum midplane runs per processor (--midplanesperproc) 2

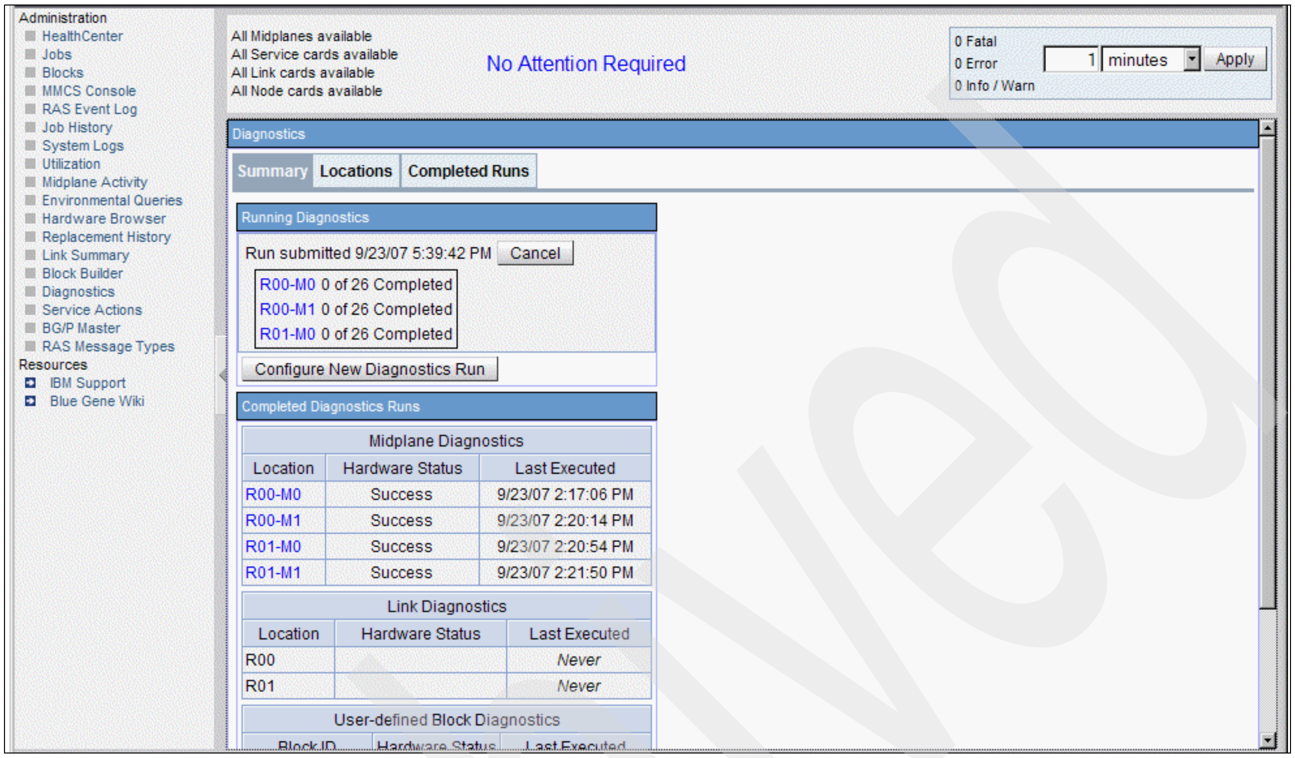
☐ Override maximum midplane runs per row (--midplanesperrow) 16

☐ Cancel jobs running on the midplane. (--killjobs)

☐ Delete the temporary block created by diagnostics at the end of the run. (--deleteblock)

Figure 9-7 Run options

After the run starts, you are returned to the Summary tab. There, you see a list of all the midplanes that are configured to run in the Running Diagnostics area (Figure 9-8). You can cancel the run at any time by clicking **Cancel** next to the start time of the run.



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

No Attention Required

0 Fatal
0 Error
0 Info / Warn

1 minutes

Run submitted 9/23/07 5:39:42 PM

R00-M0 0 of 26 Completed
R00-M1 0 of 26 Completed
R01-M0 0 of 26 Completed

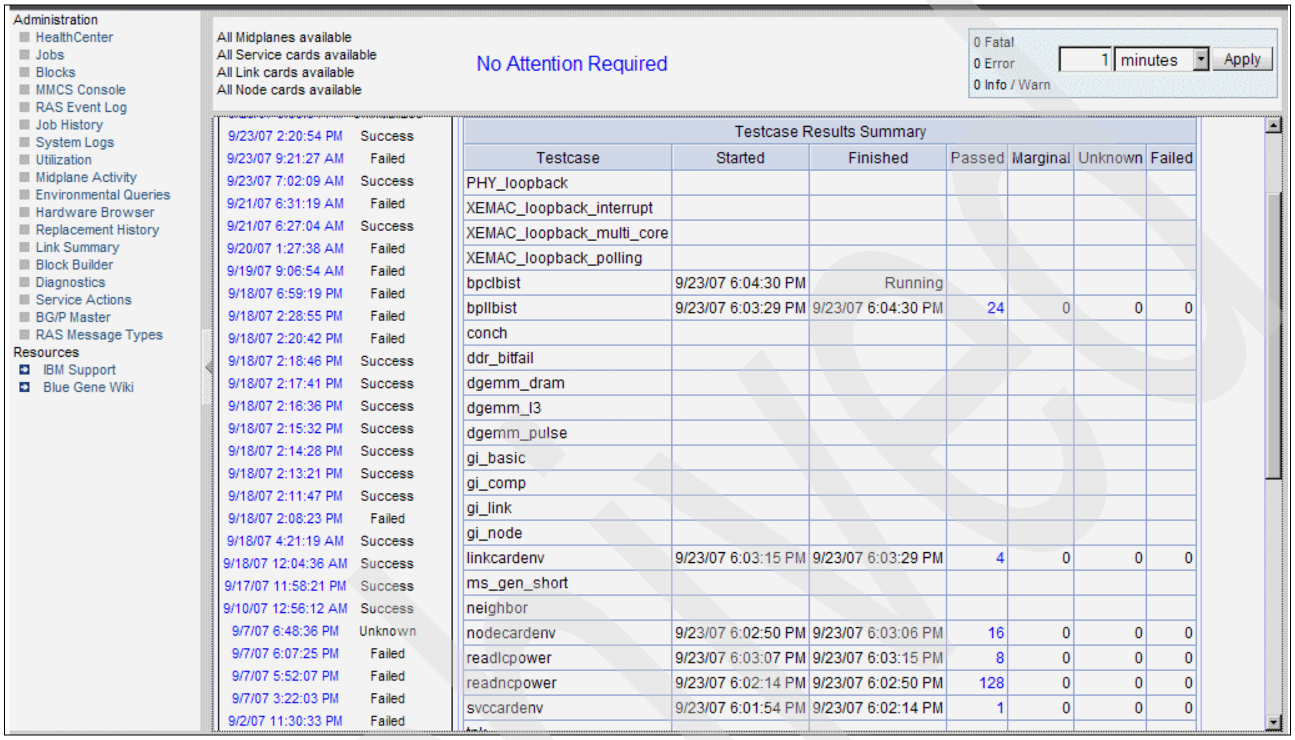
Midplane Diagnostics		
Location	Hardware Status	Last Executed
R00-M0	Success	9/23/07 2:17:06 PM
R00-M1	Success	9/23/07 2:20:14 PM
R01-M0	Success	9/23/07 2:20:54 PM
R01-M1	Success	9/23/07 2:21:50 PM

Link Diagnostics		
Location	Hardware Status	Last Executed
R00		*Never*
R01		*Never*

User-defined Block Diagnostics		
Block ID	Hardware Status	Last Executed

Figure 9-8 Diagnostic run started

Click the midplane link to view each of the test cases as well as the results that are configured to run (Figure 9-9). The number of test cases run and the status of the run are updated continually as the Diagnostic run progresses. Use the Refresh button at the bottom of the page to update the status of the run. You can set the Auto refresh feature by entering the number of minutes that you want between refreshes and then selecting the box to toggle on the check mark.



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

No Attention Required

0 Fatal
0 Error
0 Info / Warn
1 minutes

9/23/07 2:20:54 PM	Success
9/23/07 9:21:27 AM	Failed
9/23/07 7:02:09 AM	Success
9/21/07 6:31:19 AM	Failed
9/21/07 6:27:04 AM	Success
9/20/07 1:27:38 AM	Failed
9/19/07 9:06:54 AM	Failed
9/18/07 6:59:19 PM	Failed
9/18/07 2:28:55 PM	Failed
9/18/07 2:20:42 PM	Failed
9/18/07 2:18:46 PM	Success
9/18/07 2:17:41 PM	Success
9/18/07 2:16:36 PM	Success
9/18/07 2:15:32 PM	Success
9/18/07 2:14:28 PM	Success
9/18/07 2:13:21 PM	Success
9/18/07 2:11:47 PM	Success
9/18/07 2:08:23 PM	Failed
9/18/07 4:21:19 AM	Success
9/18/07 12:04:36 AM	Success
9/17/07 11:58:21 PM	Success
9/10/07 12:56:12 AM	Success
9/7/07 6:48:36 PM	Unknown
9/7/07 6:07:25 PM	Failed
9/7/07 5:52:07 PM	Failed
9/7/07 3:22:03 PM	Failed
9/2/07 11:30:33 PM	Failed

Testcase Results Summary

Testcase	Started	Finished	Passed	Marginal	Unknown	Failed
PHY_loopback						
XEMAC_loopback_interrupt						
XEMAC_loopback_multi_core						
XEMAC_loopback_polling						
bpclbist	9/23/07 6:04:30 PM	Running				
bpllbist	9/23/07 6:03:29 PM	9/23/07 6:04:30 PM	24	0	0	0
conch						
ddr_bitfail						
dgemm_dram						
dgemm_l3						
dgemm_pulse						
gi_basic						
gi_comp						
gi_link						
gi_node						
linkcardenv	9/23/07 6:03:15 PM	9/23/07 6:03:29 PM	4	0	0	0
ms_gen_short						
neighbor						
nodecardenv	9/23/07 6:02:50 PM	9/23/07 6:03:06 PM	16	0	0	0
readlcpower	9/23/07 6:03:07 PM	9/23/07 6:03:15 PM	8	0	0	0
readncpower	9/23/07 6:02:14 PM	9/23/07 6:02:50 PM	128	0	0	0
svccardenv	9/23/07 6:01:54 PM	9/23/07 6:02:14 PM	1	0	0	0

Figure 9-9 Test case details

9.3.2 Running diagnostics from the command line

You can also run diagnostics from the command line. The **`rundiags.py`** command is located in the `/bgsys/drivers/ppcfloor/bin` directory. It uses the following syntax:

```
$ /bgsys/drivers/ppcfloor/bin/rundiags.py --midplanes R00-M0,R01-M1
```

This example runs the Complete Diagnostic bucket on midplanes R00-M0 and R01-M1.

The command requires that you specify either midplane, racks, or blocks on which to run the diagnostic test cases. The command uses the following switches:

`--midplanes x,y,z...`	List of midplanes on which to run diagnostics (for example, R00-M0, R00-M1, and R01-M0)
`--racks x,y,z...`	List of racks on which to run diagnostics. (for example, R00, R01, and R10)
`--blocks x,y,z...`	List of blocks on which to run diagnostics

The list of midplanes, racks, or blocks is a comma separated list and must not contain any whitespaces. You must specify the `--blocks` switch by itself. It is not compatible with the `--rack` or `--midplanes` switches. You can use midplanes and racks in the same command, for example:

```
$ /bgsys/drivers/ppcfloor/bin/rundiags.py --midplanes midplane_list --racks
rack_list [OPTIONS ... ]
```

Table 9-2 shows the options that you can use with the **`rundiags.py`** command.

Table 9-2 The rundiags options

Option	Default	Description
`--bootoptions`	mergepers	Manually specify boot options
`--bringupoptions`	fischer_connector_clock	Manually specify bringup options
`--buckets`	Complete	Runs the specified bucket (see 9.2, “Diagnostic test cases” on page 168)
`--cloc`	Not used by default	The location string to use on connect requests
`--createxmlresult`	false	Indicates if the results.xml file should be created
`--csport`	32031	Control system port
`--csuser`	current user	Control system user
`--dbname`	bgdb0	Name of the database (see the db.properties file)
`--dbpassword`	*xxxxxx*	Database password (see db.properties file)
`--dbport`	50001	Port that DB2 is listening on
`--dbproperties`	/bgsys/local/etc/db.properties	db.properties file
`--dbschema`	bgpsysdb	Name of the database schema (see db.properties file)
`--dbuser`	bgpsysdb	Name or the database user (see db.properties file)
`--deleteblock`	false	Diagnostics will delete the blocks when done.
`--floor`	/bgsys/drivers/ppcfloor	Specify an alternate floor directory
`--forceblockfree`	false	Forces a free of the diagnostics block prior to use.
`--help`		Displays help text and exits
`--killjobs`	false	Kill any jobs or free any blocks that might prevent the run from proceeding.
`--logdirprefix`	None	Add a user-defined prefix to the log directory
`--mcserverport`	1206	mcServer port
`--midplanesperproc`	2	The number of concurrent midplane runs allowed per service node processor

Option	Default	Description
`--midplanesperrow`	16	The number of concurrent midplane runs allowed per row.
`--nodeleteblock`	true	Diagnostics will not delete the block when done.
`--nogenblock`	false	Prevents diagnostics from creating blocks automatically
`--outputdir`	/bgsys/logs	Overrides the default output directory
`--pid`	PID	Specifies the process ID to assume when generating the run ID
`--pset`	Maximum I/O to compute node ratio on the system	Overrides the default pset ratio
`--savealloutput`	false	Saves all the output generated by diagnostics
`--sn`	localhost	Name of the service node
`--stoponerror`	false	Will cause the run to abort when the first error is encountered
`--tests`	Run all tests	This option allows you to provide a list of tests to be included in the run. Can be used in conjunction with --buckets to run additional test cases.
`--testxml`	test.xml	Diagnostics tests file
`--verbosity`	0	Verbosity level of output generated by diagnostics

9.4 Preventative maintenance

We recommend that you run the diagnostics checkup bucket on each midplane at least once per month. This preventative maintenance provides coverage to determine any hardware problems while maintaining a minimum diagnostics runtime.

Note that the checkup bucket might fail for a particular hardware problem, but you can obtain more in-depth analysis by running a more complete diagnostics bucket. Run the checkup bucket to find any failing hardware, and if you need further diagnosis then run other diagnostics. The diagnostics that you will run depends highly on the specific failure.

9.5 Running diagnostics through a scheduler

To run diagnostics with a scheduler, you need to create the diagnostics blocks ahead of time. Create the diagnostics blocks on a per-midplane basis, and name them with the diagnostics naming convention `_DIAGS_`*`Rxx-Mx`*. Using this naming convention, an example diagnostics command line is as follows:

```
./rundiags.py --midplane R00-M0 --nogenblock --bucket checkup
```

You need to run the command from `/bgsys/drivers/ppcfloor/bin` by user ID bgpadmin on the service node.

9.5.1 Running diagnostics from a LoadLeveler job

Example 9-1 is a sample LoadLeveler job command file for running diagnostics. Note that the job class `diags` is defined only on the service node to guarantee that the job can be started only on the service node as required.

Example 9-1 Job command file

```
bgpadmin@bgpdd2sys1:> cat rundiag_R10-M1.cmd
# @ job_name = rundiags
# @ error = $(job_name).$(jobid).out
# @ output = $(job_name).$(jobid).out
# @ environment = COPY_ALL;
# @ wall_clock_limit = 00:30:00,00:25:00
# @ notification = error
# @ notify_user = bgpadmin
# @ job_type = bluegene
# @ bg_partition = _DIAGS_R10-M1
# @ class = diags
# @ queue
cd /bgsys/drivers/ppcfloor/bin
/bgsys/drivers/ppcfloor/bin/rundiags.py --midplane R10-M1 --test dgemm_l3
--nogenblock
```

After submitting the job command file (in Example 9-1) to LoadLeveler and after the job runs, the output shown in Example 9-2 is generated.

Example 9-2 Output file

```
bgpadmin@bgpdd2sys1:> cat rundiags.708.out
Blue Gene Diagnostics Version 2.4 Build 2 running on Linux 2.6.16.60-0.27-ppc64
ppc64, bgpdd2sys1:172.16.3.1 built on 10-16-2008 04:51:23 by bgbuild, compile 0
Blue Gene Diagnostics initializing...
Blue Gene Diagnostics starting...
Diagnostics run parameters:
        Run ID:                     0810161801126672
        Command line arguments:     --midplane R10-M1 --test dgemm_l3
--nogenblock
        Host name:                  bgpdd2sys1
        Environment:                NORMAL
        Available processors:       16
        Maximum available memory:   3.906 GiB
        Process ID:                 6672
        SN address:                 localhost
        SN control system port:     32031
        mcServer port:              1206
        SN database port:           50001
        Control system user:        bgpadmin
        Database name:              bgdb0
        Database schema:            bgpsysdb
        Database user:              bgpsysdb
        Test XML file:              tests.xml
        HUP override enabled:       true
        Output directory:           /bgsys/logs/BGP/diags/081016_180112_6672
        User specified output dir:  false
        Verbosity:                  0
        Stop on first error:        false
        Save all output:            false
        Generate block:             false
        Force block free:           false
        Default bringup options:    fischer_connector_clock
        Default boot options:       mergepers
        Floor directory:            /bgsys/drivers/ppcfloor
        Poll power regulators:      true
        Regulator polling interval: 10
        Regulator cooldown time:    60
        Check regulator levels:     true
        DRAM CE flood cancels test: true
        Midplanes:                  R10-M1
        User specified tests to run: true
        Tests to run:               dgemm_l3
Running dgemm_l3 on block _DIAGS_R10-M1. Results stored in
/bgsys/logs/BGP/diags/081016_180112_6672/dgemm_l3__DIAGS_R10-M1_180126566/
dgemm_l3 summary:
        =================================================
        Run ID:                810161801126672
        Block ID:              _DIAGS_R10-M1
        Start time:            Oct 16 18:01:26
        End time:              Oct 16 18:01:53
        Testcase:              dgemm_l3
        Passed:                516
        Marginal:              0
        Failed:                0
        Unknown:               0
        Hardware status:       success
        Internal test failure:  false
        =================================================
Diagnostics log directory: /bgsys/logs/BGP/diags/081016_180112_6672/
Blue Gene diagnostics finished.
```

9.6 Diagnostics performance considerations

The performance of the service node and the service network can directly affect the diagnostics results. Running diagnostics on too many midplanes or performing too much work on the service node during diagnostics runs can cause diagnostics timeouts, usually resulting in an UNKNOWN hardware status.

To cope with various hardware configurations and system usage scenarios, the diagnostics harness includes a small scheduler to pace the number of midplanes that are running at any given time. By default, the diagnostics harness only allows two midplanes per service node processor and 16 midplanes per row to run concurrently. Using large service nodes or reducing the amount of service node traffic during diagnostics from other midplanes (usually associated with booting other midplanes) can allow you to test more midplanes concurrently.

The Navigator Web page for starting a new diagnostics run includes two options that you can use to customize the number of concurrent midplane runs:

- `--midplanesperproc`
- `--midplanesperrow`

These options are also the command line arguments that you use to provide the same function.

The `--midplanesperproc` value describes the maximum number of midplane runs that are allowed to run in parallel per processor that is installed in the service node. This option helps prevent the service node from being overtaxed by processing that is required for diagnostics, which can cause timeouts in the tests. Thus, if you specify `--midplanesperproc 2` and the service node has six processors installed, then 12 midplane runs are allowed to run at a time if the `--midplanesperproc` value is the limiting factor.

The `--midplanesperrow` value can also limit the number of midplanes that are allowed to run at one time. This option describes the number of midplanes that are allowed to run within the same row. The row can also be a limiting factor for diagnostics in that the service network bandwidth is shared on a row-by-row basis. Spreading the diagnostics runs across the rows helps to prevent service network bottlenecks.

10

Blue Gene/P tools

This chapter describes tools that are provided with Blue Gene/P and includes sections on the Security Administration tool and the Coreprocessor tool.

10.1 Security Administration tool

The Security Administration tool provides a means of assigning authority consistently to the users who access the Blue Gene system. The tool authorizes users to various predetermined functions on the system by adding their profile to a selected group. The predefined groups are:

- bgpuser
- bgpdeveloper
- bgpservice
- bgpadmin

The groups are created on the service node when the Blue Gene/P system is installed. You can edit the program to define the groups differently.

Existing Linux users are added to groups by running the bguser.pl utility. The command uses the following syntax:

```
./bguser.pl [options]
```

Options are:

- `--user userName`
- `--role [user/developer/service/admin]`

Table 10-1 lists the available roles and capabilities that are allowed when a user is added using the tool.

Table 10-1 The bguser roles

Role	Capability
user	► Submit jobs through mpirun, High Performance Computing (HPC) ► **`submit`** command, HTC ► Read access to small amount of data (job/block status) on service node, through Navigator ► Access to the front end nodes ► Complete access compilers, tool chain, and so forth for development on the front end nodes
developer	► Submit jobs through mpirun, HPC ► **`submit`** command, HTC ► Read access to some data (job/block status) on service node, through Navigator ► Controlled and limited access to service node, which requires user ID on SN ► Does not have root access but has elevated privileges ► Complete access compilers, tool chain, and so forth for development on the front end nodes ► Debugging with coreprocessor
admin	► Complete access to Blue Gene/P functions on the service node and front end nodes
service (IBM)	► Access to required debug tools, system logs, read access to database

10.2 Coreprocessor tool

The Coreprocessor tool is a basic parallel debugger that enables parallel debug of problems at all levels (hardware, kernel, and application). The Coreprocessor tool uses the low-level hardware JTAG interface to read and organize hardware information, including instruction address registers (IAR), general purpose registers (GPR), special purpose registers (SPR), and device control registers (DCR). It can process compute node core files, and it can also connect to a running job.

The Coreprocessor tool levies no dependencies on the application code that is running on the node (no special calls that need to be made, libraries to link in, and so on). It can sort nodes based on their stack traceback and kernel status, which can help isolate a failing or problem node quickly. It also supports stack dumping on a per processor basis.

Most likely, you can use the tool for the following functions:

- To examine compute node core files. For performance and disk space reasons, these core files are simple text files, and their format is not understood by all debuggers. The Coreprocessor tool points out the node that is acting in a suspicious manner and that probably caused the partition to dump.
- When your usual debugger cannot connect to the compute nodes. Other debuggers are not useful when a node is non-responsive. However, because the Coreprocessor tool uses the JTAG connection to a node to collect debug information, it does not require a functional connection to a compute node through an I/O node.

Using the Coreprocessor tool provide the following advantages:

- The operating system can be completely dead and can still handle debug
- Quick isolation of nodes that are abnormal
- Scales to a full Blue Gene/P system

With the Coreprocessor tool, you cannot set breakpoints, and there is no visibility. However, it does provide the instruction address and function name. It also provides the line number when the code has been compiled with the debug option.

Dependencies

The Coreprocessor tool uses the following PERL modules:

- cwd
- Compress::Zlib
- IO::Socket
- OpenGL (for 3D Map function only)
- Tk

Note: Tk is not usually installed with the operating system. One way to install it is to run the following command:

```
sudo perl -MCPAN -e “install Tk” \;
```

Answer *no* at the prompt to let it autoconfigure.

10.2.1 Starting the Coreprocessor tool

The Coreprocessor tool is initiated from a VNC session. From the command line (VNC), enter the following command:

```
/bgsys/drivers/ppcfloor/tools/coreprocessor/coreprocessor.pl
```

The tool does not require parameters to start. Table 10-2 provides a list of options that you can use when starting the tool.

Table 10-2 Options to use when starting the tool

Option	Description
`-b`	Specifies the ELF image to load for compute nodes. Multiple ELF files can be specified by concatenating them with a colon (:). Typically the user's application is specified here. Example: `-b=/bguser/username/hello_world.elf:/bgsys/drivers/ppcfloor/cnk/bgp_kernel.cnk.elf`
`-lb`	Specifies the ELF image to load for I/O nodes. Typically the vmlinux.elf file. Example: `-lb=/bgsys/drivers/ppcfloor/boot/vmlinuz-2.6.19`

`-c`	Specifies the path to the directory containing existing core files. Example: `-c=/bgusr/username/`
	You can use the following options:
	`-minicore` Specifies the lowest numbered core file to load. The default is 0. Example: `-minicore=75`
	`-maxcore` Specifies the highest numbered core file to load. The default is 131072. Example: `-maxcore=65536`

`-a`	Connects to MMCS server and attaches to the specified, running block. Example: `-a=RMP04My010523010`
	You can use the following options:
	`-host` Host name or IP address for the MMCS server. Typically this is either localhost or the service node's IP address. Example: `-host=bgpServiceNode.YourDomain.com`
	`-port` The TCP port that the MMCS server is listening on for console connections. Example: `-port=32031`
	`-user` The user name or owner of the mmcs block. Example: `-user=bgluser`
	`-numbadframes` Number of the topmost stack frames to remove from collection. This is useful if the application under debug is doing some customer assembly. Under these conditions the application might not strictly adhere to the PowerPC ABI governing stack frame usage. Example: `-numbadframes=1`
	`-numthreads` Number of mmcs threads that the Coreprocessor tool will use to parallelize data extraction. Example `-numthreads=8`

`-s`	Loads the specified Coreprocessor snapshot file for post-failure analysis. Example: `-s=/bgusr/username/linpack.snapshot`
`-templates`	Specifies the directory for saved memory templates. Example: `-templates=/bgusr/username/mytemplates/.`

Option	Description

`-nogui`	GUI-less mode. Coreprocessor tool processes the data and terminates. Example: `-nogui`
	You can use these options:
	`-mode` Specifies the type of analysis to perform when Coreprocessor is running in GUI-less mode. Valid modes include: Condensed, Detailed, Survey, Instruction, Kernel, Processor, DCR, PPC, Ungroup_trace, Ungroup, and Neighbor. Example: `-mode=Processor`
	`-register` Specifies the PPC or DCR register when using the PPC or DCR modes. Example: `-register=GPR3`
	`-snapshot` Fetch failure data and save to snapshot file. Example: `-snapshot=/tmp/hang_issue1234`

`-h`	Displays help text. Example: -h

Figure 10-1 shows the graphical user interface (GUI) Coreprocessor tool started in a VNC session.

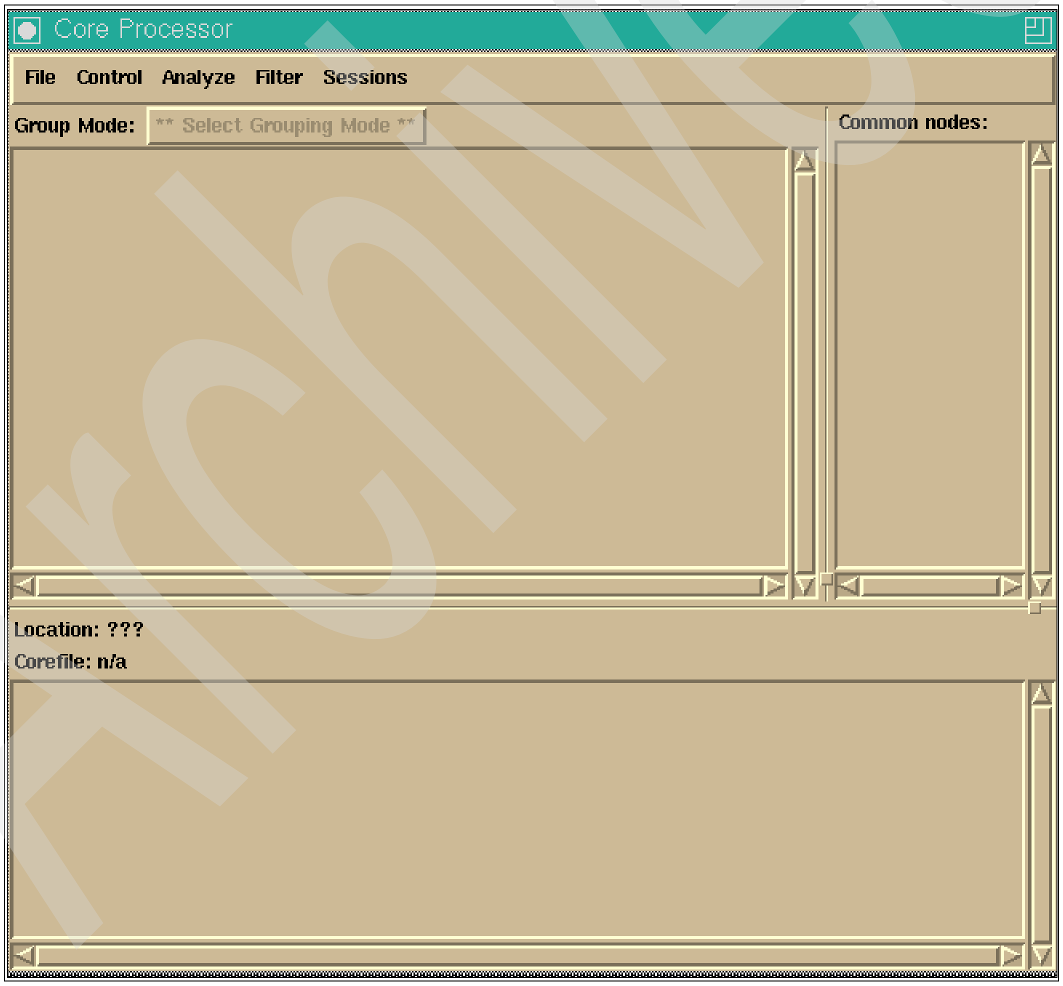


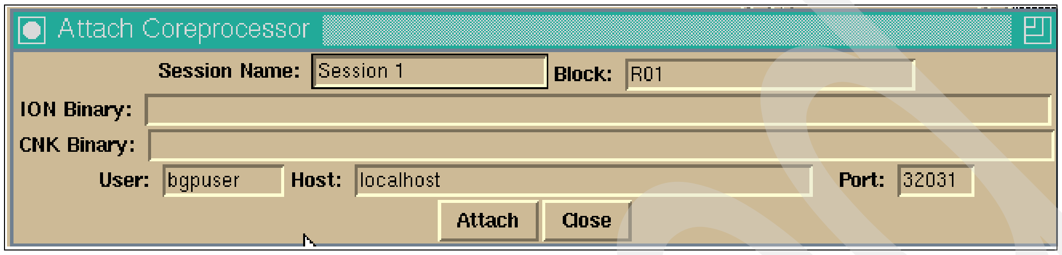
Figure 10-1 Coreprocessor GUI

10.2.2 Debugging live compute node problems

When debugging an unresponsive block, you are looking usually for one or a small subset of compute nodes that are doing something different from the majority of compute nodes.

To do live debug on compute nodes:

1. Start the Coreprocessor GUI, and then select **File ∅ Attach To Block**. The window shown in Figure 10-2 opens.

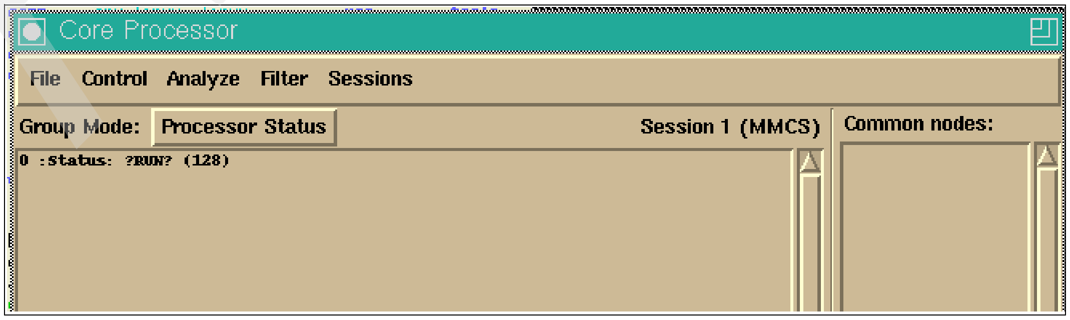


Session Name: Session 1
Block: R01
ION Binary:
CNK Binary:
User: bgpuser
Host: localhost
Port: 32031

Figure 10-2 Coreprocessor attach window

2. Supply the following information:
   - Session Name. You can run more than one session at a time, so this is used to distinguish between multiple sessions.
   - Block name.
   - CNK binary (with path). To see both your application and the Compute Node Kernel in the stack, specify the Compute Node Kernel Image and your application binary separated by a colon. For our example, we used:

     ```
     /bgsys/drivers/ppcfloor/boot/cnk:/bgusr/pallas/PMB-MPI1
     ```
   - You can specify ION binary as well, but refer to 10.2.4, "Debugging live I/O node problems" on page 197.
   - User name or owner of the MMCS block.
   - TCP port on which MMCS server is listening for console connections (probably 32031).
   - Host name or TCP/IP address for MMCS server. Typically this is *localhost* or the service node's TCP/IP address.
3. Then click **Attach**. At this point, you have not yet affected the state of the processors. Choose **Select Grouping Mode ∅ Processor Status**. Notice the text in the upper-left pane (Figure 10-3). The Coreprocessor tool posts the status *?RUN?* because it does not yet know the state of the processors. The number *128* is the number of nodes in the block that are in that state. The number in parenthesis always indicates the number of nodes that share whatever attribute is displayed on the line (the processor state in this case).



Group Mode: Processor Status
Session 1 (MMCS)

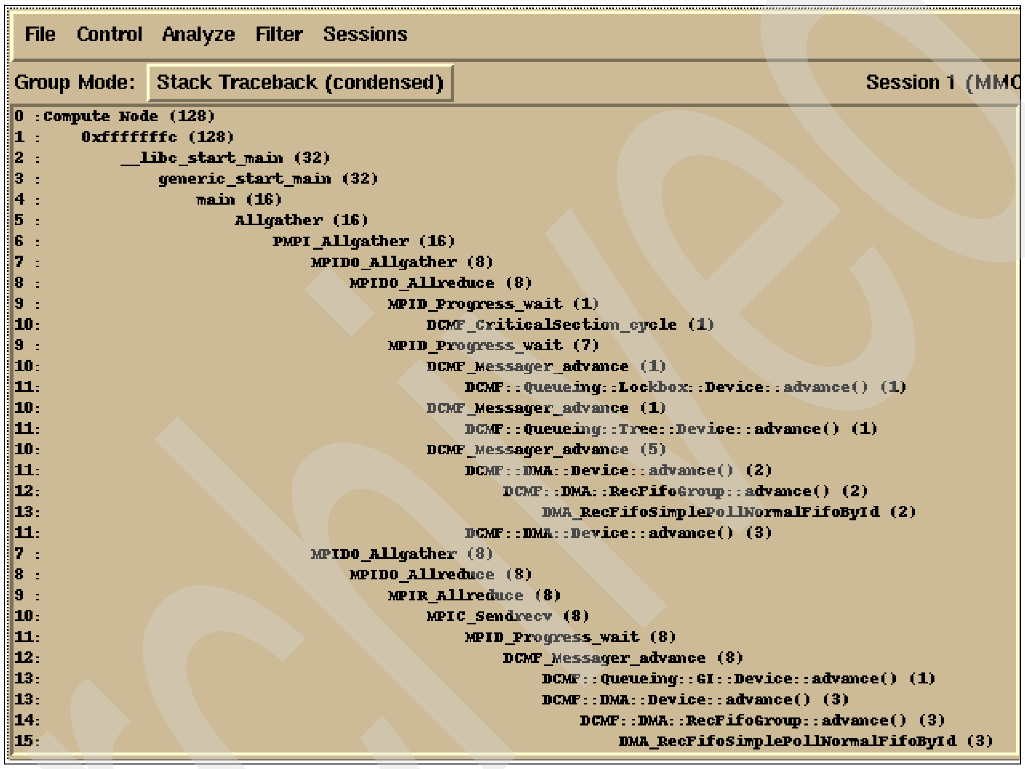
```
0 :status: ?RUN? (128)
```

Figure 10-3 Processors in Run status

4. Back at the main window, click **Select Grouping Mode**. When you choose one of the Stack Traceback options, the Coreprocessor tool halts all the compute node processor cores and displays the requested information. Choose each of the options on that menu in turn so you can see the variety of data formats that are available.

Stack Traceback (condensed)

In the condensed version of Stack Traceback, data from all nodes is captured. The unique instruction addresses per stack frame are grouped and displayed, except that the last stack frame is grouped based on the function name (not the IAR), as shown in Figure 10-4. This mode is normally the most useful mode for debug.

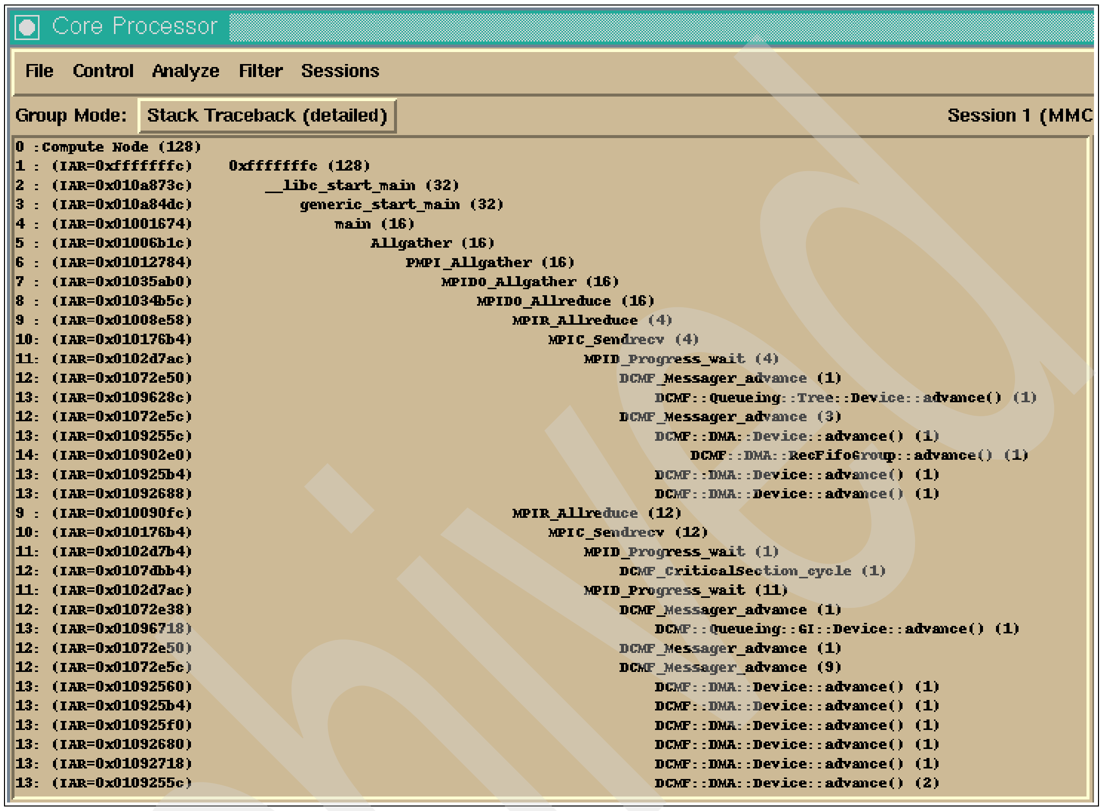


```
0 :Compute Node (128)
1 :    0xfffffffc (128)
2 :        __libc_start_main (32)
3 :            generic_start_main (32)
4 :                main (16)
5 :                    Allgather (16)
6 :                        PMPI_Allgather (16)
7 :                            MPIDO_Allgather (8)
8 :                                MPIDO_Allreduce (8)
9 :                                    MPID_Progress_wait (1)
10:                                        DCMF_CriticalSection_cycle (1)
9 :                                    MPID_Progress_wait (7)
10:                                        DCMF_Messager_advance (1)
11:                                            DCMF::Queueing::Lockbox::Device::advance() (1)
10:                                        DCMF_Messager_advance (1)
11:                                            DCMF::Queueing::Tree::Device::advance() (1)
10:                                        DCMF_Messager_advance (5)
11:                                            DCMF::DMA::Device::advance() (2)
12:                                                DCMF::DMA::RecFifoGroup::advance() (2)
13:                                                    DMA_RecFifoSimplePollNormalFifoById (2)
11:                                            DCMF::DMA::Device::advance() (3)
7 :                            MPIDO_Allgather (8)
8 :                                MPIDO_Allreduce (8)
9 :                                    MPIR_Allreduce (8)
10:                                        MPIC_Sendrecv (8)
11:                                            MPID_Progress_wait (8)
12:                                                DCMF_Messager_advance (8)
13:                                                    DCMF::Queueing::GI::Device::advance() (1)
13:                                                    DCMF::DMA::Device::advance() (3)
14:                                                        DCMF::DMA::RecFifoGroup::advance() (3)
15:                                                            DMA_RecFifoSimplePollNormalFifoById (3)
```

Figure 10-4 Stack Traceback (Condensed)

Stack Traceback (detailed)

In Stack Traceback, detailed data from all nodes are captured (see Figure 10-5). The unique instruction addresses per stack frame are grouped and displayed. The IAR at each stack frame is also displayed.

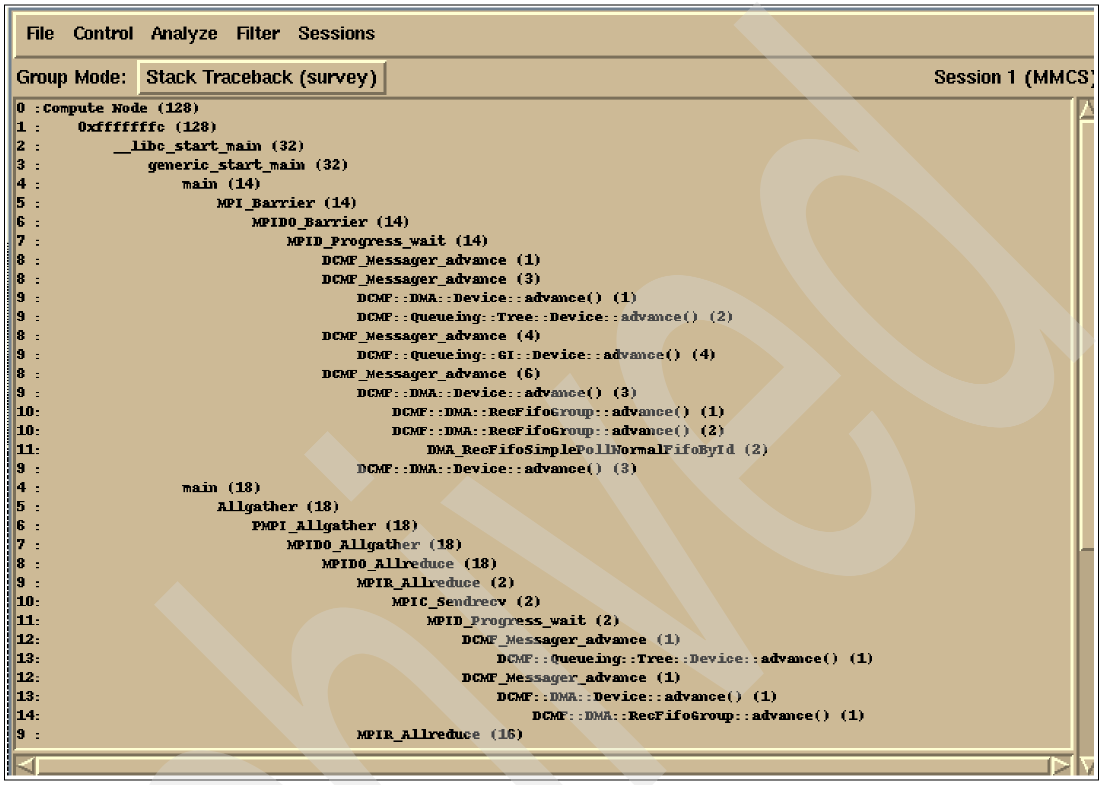


```
0 :Compute Node (128)
1 : (IAR=0xfffffffc)    0xfffffffc (128)
2 : (IAR=0x010a873c)        __libc_start_main (32)
3 : (IAR=0x010a84dc)            generic_start_main (32)
4 : (IAR=0x01001674)                main (16)
5 : (IAR=0x01006b1c)                    Allgather (16)
6 : (IAR=0x01012784)                        PMPI_Allgather (16)
7 : (IAR=0x01035ab0)                            MPIDO_Allgather (16)
8 : (IAR=0x01034b5c)                                MPIDO_Allreduce (16)
9 : (IAR=0x01008e58)                                    MPIR_Allreduce (4)
10: (IAR=0x010176b4)                                        MPIC_Sendrecv (4)
11: (IAR=0x0102d7ac)                                            MPID_Progress_wait (4)
12: (IAR=0x01072e50)                                                DCMF_Messager_advance (1)
13: (IAR=0x0109628c)                                                    DCMF::Queueing::Tree::Device::advance() (1)
12: (IAR=0x01072e5c)                                                DCMF_Messager_advance (3)
13: (IAR=0x0109255c)                                                    DCMF::DMA::Device::advance() (1)
14: (IAR=0x010902e0)                                                        DCMF::DMA::RecFifoGroup::advance() (1)
13: (IAR=0x010925b4)                                                    DCMF::DMA::Device::advance() (1)
13: (IAR=0x01092688)                                                    DCMF::DMA::Device::advance() (1)
9 : (IAR=0x010090fc)                                    MPIR_Allreduce (12)
10: (IAR=0x010176b4)                                        MPIC_Sendrecv (12)
11: (IAR=0x0102d7b4)                                            MPID_Progress_wait (1)
12: (IAR=0x0107dbb4)                                                DCMF_CriticalSection_cycle (1)
11: (IAR=0x0102d7ac)                                            MPID_Progress_wait (11)
12: (IAR=0x01072e38)                                                DCMF_Messager_advance (1)
13: (IAR=0x01096718)                                                    DCMF::Queueing::GI::Device::advance() (1)
12: (IAR=0x01072e50)                                                DCMF_Messager_advance (1)
12: (IAR=0x01072e5c)                                                DCMF_Messager_advance (9)
13: (IAR=0x01092560)                                                    DCMF::DMA::Device::advance() (1)
13: (IAR=0x010925b4)                                                    DCMF::DMA::Device::advance() (1)
13: (IAR=0x010925f0)                                                    DCMF::DMA::Device::advance() (1)
13: (IAR=0x01092680)                                                    DCMF::DMA::Device::advance() (1)
13: (IAR=0x01092718)                                                    DCMF::DMA::Device::advance() (1)
13: (IAR=0x0109255c)                                                    DCMF::DMA::Device::advance() (2)
```

Figure 10-5 Stack Traceback (detailed)

Stack Traceback (survey)

Stack Traceback (survey) is a quick but potentially inaccurate mode. IARs are initially captured, and stack data is collected for each node from a group of nodes that contain the same IAR. The stack data fetched for that one node is then applied to all nodes with the same IAR. Figure 10-6 shows an example of survey mode.



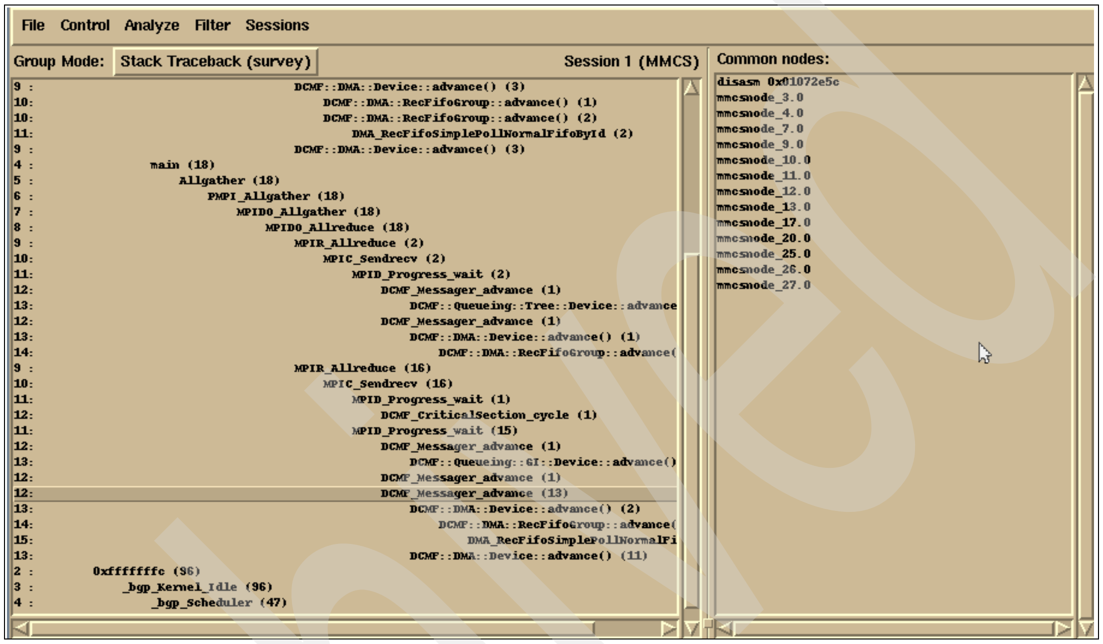
Group Mode: Stack Traceback (survey)   Session 1 (MMCS)

```
0 :Compute Node (128)
1 :    0xfffffffc (128)
2 :        __libc_start_main (32)
3 :            generic_start_main (32)
4 :                main (14)
5 :                    MPI_Barrier (14)
6 :                        MPIDO_Barrier (14)
7 :                            MPID_Progress_wait (14)
8 :                                DCMF_Messager_advance (1)
8 :                                DCMF_Messager_advance (3)
9 :                                    DCMF::DMA::Device::advance() (1)
9 :                                    DCMF::Queueing::Tree::Device::advance() (2)
8 :                                DCMF_Messager_advance (4)
9 :                                    DCMF::Queueing::GI::Device::advance() (4)
8 :                                DCMF_Messager_advance (6)
9 :                                    DCMF::DMA::Device::advance() (3)
10:                                        DCMF::DMA::RecFifoGroup::advance() (1)
10:                                        DCMF::DMA::RecFifoGroup::advance() (2)
11:                                            DMA_RecFifoSimplePollNormalFifoById (2)
9 :                                    DCMF::DMA::Device::advance() (3)
4 :                main (18)
5 :                    Allgather (18)
6 :                        PMPI_Allgather (18)
7 :                            MPIDO_Allgather (18)
8 :                                MPIDO_Allreduce (18)
9 :                                    MPIR_Allreduce (2)
10:                                        MPIC_Sendrecv (2)
11:                                            MPID_Progress_wait (2)
12:                                                DCMF_Messager_advance (1)
13:                                                    DCMF::Queueing::Tree::Device::advance() (1)
12:                                                DCMF_Messager_advance (1)
13:                                                    DCMF::DMA::Device::advance() (1)
14:                                                        DCMF::DMA::RecFifoGroup::advance() (1)
9 :                                    MPIR_Allreduce (16)
```

Figure 10-6   Stack Traceback (survey)

The following points can help you use the tool more effectively:

- The number at the far left, before the colon, indicates the depth within the stack.
- The number in parentheses at the end of each line indicates how many of the nodes share that same stack frame.
- If you click any line in the stack dump, the upper-right pane (labeled Common nodes) shows the list of nodes that share that stack frame. See Figure 10-7.



```
9 :                            DCMF::DMA::Device::advance() (3)
10:                               DCMF::DMA::RecFifoGroup::advance() (1)
10:                               DCMF::DMA::RecFifoGroup::advance() (2)
11:                                  DMA_RecFifoSimplePollNormalFifoById (2)
9 :                            DCMF::DMA::Device::advance() (3)
4 :              main (18)
5 :                 Allgather (18)
6 :                    PMPI_Allgather (18)
7 :                       MPIDO_Allgather (18)
8 :                          MPIDO_Allreduce (18)
9 :                             MPIR_Allreduce (2)
10:                                MPIC_Sendrecv (2)
11:                                   MPID_Progress_wait (2)
12:                                      DCMF_Messager_advance (1)
13:                                         DCMF::Queueing::Tree::Device::advance
12:                                      DCMF_Messager_advance (1)
13:                                         DCMF::DMA::Device::advance() (1)
14:                                            DCMF::DMA::RecFifoGroup::advance(
9 :                             MPIR_Allreduce (16)
10:                                MPIC_Sendrecv (16)
11:                                   MPID_Progress_wait (1)
12:                                      DCMF_CriticalSection_cycle (1)
11:                                   MPID_Progress_wait (15)
12:                                      DCMF_Messager_advance (1)
13:                                         DCMF::Queueing::GI::Device::advance()
12:                                      DCMF_Messager_advance (1)
12:                                      DCMF_Messager_advance (13)
13:                                         DCMF::DMA::Device::advance() (2)
14:                                            DCMF::DMA::RecFifoGroup::advance(
15:                                               DMA_RecFifoSimplePollNormalFi
13:                                         DCMF::DMA::Device::advance() (11)
2 :        0xfffffffc (96)
3 :           _bgp_Kernel_Idle (96)
4 :              _bgp_Scheduler (47)
```

```
disasm 0x01072e5c
mmcsnode_3.0
mmcsnode_4.0
mmcsnode_7.0
mmcsnode_9.0
mmcsnode_10.0
mmcsnode_11.0
mmcsnode_12.0
mmcsnode_13.0
mmcsnode_17.0
mmcsnode_20.0
mmcsnode_25.0
mmcsnode_26.0
mmcsnode_27.0
```

Figure 10-7 Stack Traceback common nodes

- When you click one of the stack frames and then select **Run** from the Control pull-down, that action is performed for all nodes sharing that stack frame. A new Processor Status summary is displayed. If you chose a Stack Traceback option again, the running processors are halted and the stacks are refetched.
- You can hold down the Shift key and click several stack frames if you want to control all procedures that are at a range of stack frames.
- From the Filter pull-down, you can select Create Filter from Group Selection. This adds a filter with the name you specify in the Filter pull-down. When the box for your filter is highlighted, only the data for those processors is displayed in the upper left window. You can create several filters if you want.
- Set Group Mode to Ungrouped or Ungrouped with Traceback to control one processor at a time.

10.2.3 Saving your information

To save your Traceback information, select **File ∅ Save Traceback** to save the current contents of the upper-left pane to a file of your choosing.

To gain more complete data, click **File ∅ Take Snapshot™**. Notice that you then have two sessions to choose from on the Sessions pull-down. The original session is MMCS and the second one is SNAP. The snapshot is just what the name implies—a picture of the debug session at a particular point. Notice that you cannot start or stop the processors from the snapshot session. You can choose **File ∅ Save Snapshot** to save the snapshot to a file. If you are sending data to IBM for debug, Save Snapshot is a better choice than Save Traceback because the snapshot includes objdump data.

If you choose **File ∅ Quit** and there are processors halted, you are given an option to restart them before quitting.

10.2.4 Debugging live I/O node problems

It is possible to debug the I/O nodes as well as compute nodes, but you normally want to avoid doing so. Collecting data causes the processor to be stopped, and stopping the I/O node processors can cause problems with your file system. In addition, the compute nodes will not be able to communicate with the I/O nodes. If you want to do I/O node debug, you must specify the ION binary when you do the **File ∅ Attach** to block the pop-up window and choose Debug IONodes from the Filter pull-down menu.

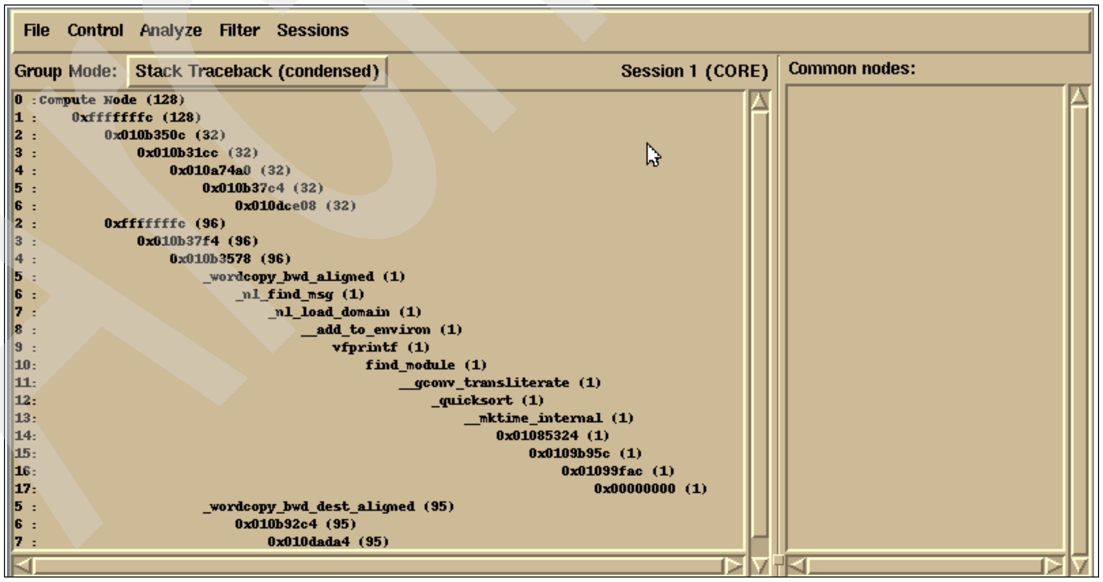
10.2.5 Debugging core files

To work with core files, select **File ∅ Load Core**. In the window that opens, specify the following:

- The location of the CNK binary or binaries
- The core files location
- The lowest and highest-numbered core files that you want to work with. (The default is all available core files.)

Click **Load Cores** when you are done.

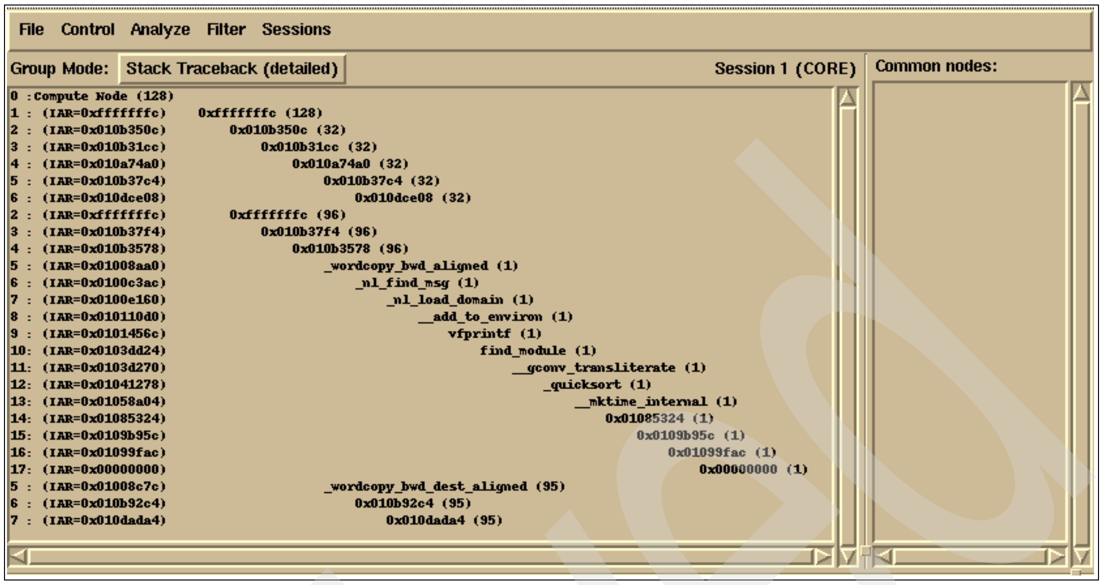
The same Grouping Modes are available for core file debug as for live debug. Figure 10-8 shows an output example of the Condensed Stack Traceback options from a core file. Condensed mode is the easiest format to work with.



```
0 :Compute Node (128)
1 :    0xfffffffc (128)
2 :        0x010b350c (32)
3 :            0x010b31cc (32)
4 :                0x010a74a0 (32)
5 :                    0x010b37c4 (32)
6 :                        0x010dce08 (32)
2 :        0xfffffffc (96)
3 :            0x010b37f4 (96)
4 :                0x010b3578 (96)
5 :                    _wordcopy_bwd_aligned (1)
6 :                        _nl_find_msg (1)
7 :                            _nl_load_domain (1)
8 :                                __add_to_environ (1)
9 :                                    vfprintf (1)
10:                                        find_module (1)
11:                                            __gconv_transliterate (1)
12:                                                _quicksort (1)
13:                                                    __mktime_internal (1)
14:                                                        0x01085324 (1)
15:                                                            0x0109b95c (1)
16:                                                                0x01099fac (1)
17:                                                                    0x00000000 (1)
5 :                    _wordcopy_bwd_dest_aligned (95)
6 :                        0x010b92c4 (95)
7 :                            0x010dada4 (95)
```

Figure 10-8 Core file condensed stack trace

Figure 10-9 shows the detailed version of the same trace.

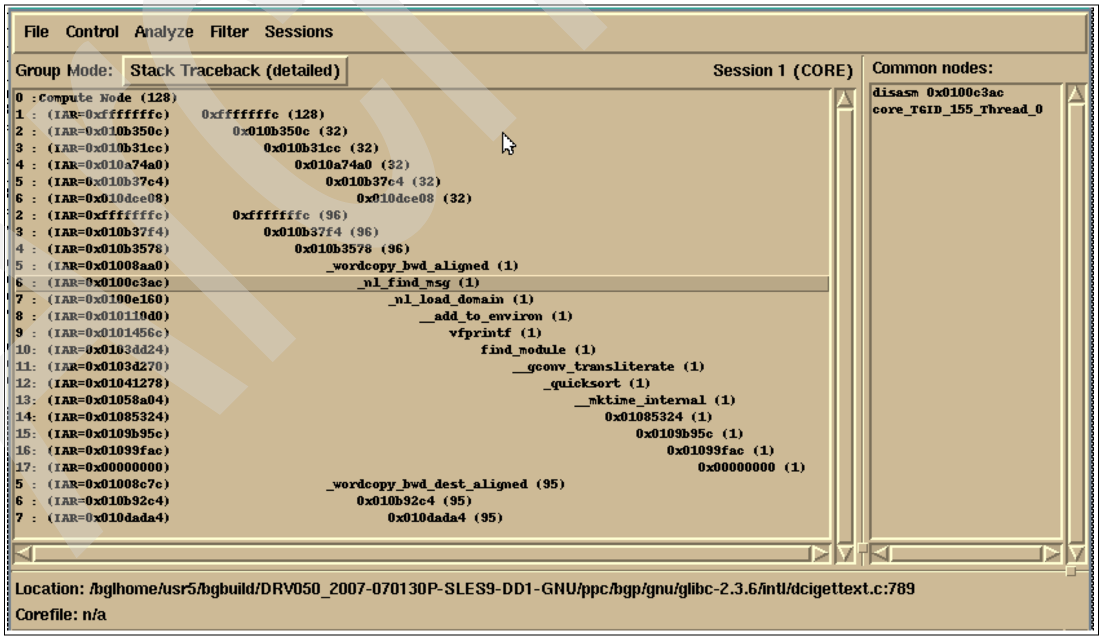


```
0 :Compute Node (128)
1 : (IAR=0xfffffffc)    0xfffffffc (128)
2 : (IAR=0x010b350c)        0x010b350c (32)
3 : (IAR=0x010b31cc)            0x010b31cc (32)
4 : (IAR=0x010a74a0)                0x010a74a0 (32)
5 : (IAR=0x010b37c4)                    0x010b37c4 (32)
6 : (IAR=0x010dce08)                        0x010dce08 (32)
2 : (IAR=0xfffffffc)        0xfffffffc (96)
3 : (IAR=0x010b37f4)            0x010b37f4 (96)
4 : (IAR=0x010b3578)                0x010b3578 (96)
5 : (IAR=0x01008aa0)                    _wordcopy_bwd_aligned (1)
6 : (IAR=0x0100c3ac)                        _nl_find_msg (1)
7 : (IAR=0x0100e160)                            _nl_load_domain (1)
8 : (IAR=0x010110d0)                                __add_to_environ (1)
9 : (IAR=0x0101456c)                                    vfprintf (1)
10: (IAR=0x0103dd24)                                        find_module (1)
11: (IAR=0x0103d270)                                            __gconv_transliterate (1)
12: (IAR=0x01041278)                                                _quicksort (1)
13: (IAR=0x01058a04)                                                    __mktime_internal (1)
14: (IAR=0x01085324)                                                        0x01085324 (1)
15: (IAR=0x0109b95c)                                                            0x0109b95c (1)
16: (IAR=0x01099fac)                                                                0x01099fac (1)
17: (IAR=0x00000000)                                                                    0x00000000 (1)
5 : (IAR=0x01008c7c)                    _wordcopy_bwd_dest_aligned (95)
6 : (IAR=0x010b92c4)                        0x010b92c4 (95)
7 : (IAR=0x010dada4)                            0x010dada4 (95)
```

Figure 10-9   Core file detailed stack trace

The Survey option is not as useful for core files because speed is not such a concern.

When you select a stack frame in the traceback output, two additional pieces of information are displayed (Figure 10-10). The core files that share that stack frame are displayed in the Common nodes pane; the Location field under the traceback pane displays the location of that function and the line number represented by the stack frame. If you select one of those core files in the Common nodes pane, the contents of that core file are displayed in the bottom pane.



```
0 :Compute Node (128)
1 : (IAR=0xfffffffc)    0xfffffffc (128)
2 : (IAR=0x010b350c)        0x010b350c (32)
3 : (IAR=0x010b31cc)            0x010b31cc (32)
4 : (IAR=0x010a74a0)                0x010a74a0 (32)
5 : (IAR=0x010b37c4)                    0x010b37c4 (32)
6 : (IAR=0x010dce08)                        0x010dce08 (32)
2 : (IAR=0xfffffffc)        0xfffffffc (96)
3 : (IAR=0x010b37f4)            0x010b37f4 (96)
4 : (IAR=0x010b3578)                0x010b3578 (96)
5 : (IAR=0x01008aa0)                    _wordcopy_bwd_aligned (1)
6 : (IAR=0x0100c3ac)                        _nl_find_msg (1)
7 : (IAR=0x0100e160)                            _nl_load_domain (1)
8 : (IAR=0x010110d0)                                __add_to_environ (1)
9 : (IAR=0x0101456c)                                    vfprintf (1)
10: (IAR=0x0103dd24)                                        find_module (1)
11: (IAR=0x0103d270)                                            __gconv_transliterate (1)
12: (IAR=0x01041278)                                                _quicksort (1)
13: (IAR=0x01058a04)                                                    __mktime_internal (1)
14: (IAR=0x01085324)                                                        0x01085324 (1)
15: (IAR=0x0109b95c)                                                            0x0109b95c (1)
16: (IAR=0x01099fac)                                                                0x01099fac (1)
17: (IAR=0x00000000)                                                                    0x00000000 (1)
5 : (IAR=0x01008c7c)                    _wordcopy_bwd_dest_aligned (95)
6 : (IAR=0x010b92c4)                        0x010b92c4 (95)
7 : (IAR=0x010dada4)                            0x010dada4 (95)
```

```
disasm 0x0100c3ac
core_TGID_155_Thread_0
```

Location: /bglhome/usr5/bgbuild/DRV050_2007-070130P-SLES9-DD1-GNU/ppc/bgp/gnu/glibc-2.3.6/intl/dcigettext.c:789
Corefile: n/a

Figure 10-10   Core files common nodes

11

Service Actions

Preparing the Blue Gene/P hardware for service or for a system power cycle is a normal part of the System Administrator's duties. In this chapter, we describe Service Actions and discuss how to prepare the hardware for service and how to cycle power.

11.1 Service Actions overview

There are occasions when you need to cycle power or replace hardware on your Blue Gene/P system. When you need to perform service on the Blue Gene/P hardware, you need to prepare the hardware by generating a *Service Action*.

You can implement Service Actions using one of the following methods:

- Through the Blue Gene Navigator
- Directly on the service node's command line

You can specify the following hardware in a Service Action:

- Single Node card
- Multiple Node cards on a midplane
- Link card
- Any or all cards in a midplane (except the Service card)
- Fan Module
- Bulk Power Module
- Rack (allows the Service card to be removed)
- Clock card

As a general rule, if you can remove the hardware with your fingers, then you can simply generate a Service Action for that location prior to removing the part. If, however, you need to use tools to perform the service, then you need to turn off the power to the rack before removing it.

Attention: At least one functioning Bulk Power Module (BPM) must be plugged in at all times during a Service Action so that five volt (5V) persistent power is maintained to the clock card. If the power to the rack is switched off, the clock cards lose persistent power.

When you initiate a Service Action, the following internal procedures and checks occur, which are common to most hardware prior to preparing it for service:

1. A check is done to ensure that there are no conflicting Service Actions in progress.
2. A Reliability, Availability, and Serviceability (RAS) event is written to the event logs stating that a Service Action has started on a specific piece of hardware along with the name of the person that initiated the action.
3. Jobs that are using affected hardware are ended, and blocks are freed.
4. Job schedulers are prevented from allocating resources that are marked for Service Actions.
5. Vital Product Data (VPD) is written to the database.
6. Some hardware communicates to the Service card through an Ethernet port on the midplane. When you initiate a Service Action on that type of hardware, its Ethernet port on the Service card's switch is disabled.
7. The hardware's status is updated in the database.
8. A RAS event is posted indicating that the hardware is ready for service.

Service Action logs

The Service Action logs are stored in the /bgsys/logs/BGP directory. You can access the logs from the command line or from the System Logs link in Blue Gene Navigator. The naming format uses the following syntax:

```
<service action>-location-timestamp.log
```

In this command, `<service action>` correlates to the **`Servicexxx`** command that was used to initiate the action.

For example, a Service Action started for node card R01-M0-N04 on 07/06/2007 at 13:32:25 is named *ServiceNodeCard-R01-M0-N04-2007-07-06-13:32:25.log*. When that Service Action is ended, you see another log with a similar name, but the time stamp will be different. If the Service Action from the previous example completes after 11 minutes, the log file name is *ServiceNodeCard-R01-M0-N04-2007-07-06-13:43:25.log*.

Tip: The BGPSERVICEACTION table also contains the names of the logs in the LOGFILENAMEPREPAREFORSERVICE and LOGFILENAMEENDSERVICEACTION fields. The following is a simply query to retrieve the information (where *xxx* indicates the Service Action ID number):

```
db2 "select * from bgpserviceaction where id = xxx”
```

You can replace the asterisk (*) with either of the field names to return the location of that specific file.

11.2 Preparing the hardware for service

As mentioned previously, you can initiate a Service Action from the command line or from the Navigator. The only difference between the two methods is that you cannot supply the optional parameters when submitting the Service command from the Navigator. In this section, we detail the steps for both methods.

Restriction: Service cards maintain power during a Service Action and require special handling. In order to safely remove a service card you must power off the entire rack. To prepare the rack for a power down use either the ServiceClockCard or ServiceRack command. If the rack’s clock card is part of the clock domain use the ServiceClockCard command. Use the ServiceRack command if the rack’s clock card is not being used to transmit clock signals.

11.2.1 Command line Service Actions

The commands used to start, end, and close Service Actions are located in the `/bgsys/drivers/ppcfloor/bareMetal` directory. The commands are:

- **`ServiceBulkPowerModule`**
- **`ServiceFanModule`**
- **`ServiceLinkCard`**
- **`ServiceMidplane`**
- **`ServiceNodeCard`**
- **`ServiceRack`**
- **`ServiceClockCard`**

Important: ServiceRack and ServiceClockCard require that you have physical access to the rack to recover. To end these Service Actions, you must cycle power on the rack or reseat one of the BPMs to bring the rack back online.

You need to qualify each of the commands with a location. For example, the **`ServiceNodeCard`** command requires that you give the location of a specific Node card, such as R00-M0-N04, or all the Node cards in a midplane, R00-M0-N. In addition to the location, you also need to specify the action. The list of possible actions are:

`Prepare`	Prepare the card for service
`End`	End the Service Action and make the card functional
`Close`	Force an existing open or prepared Service Action to the closed state

There are several optional parameters that can be supplied:

`--user` *`<userID>`*	The user name to be specified when connecting to mcServer. If a user is not specified, the default is the user who issued the command.
`--diags<test value>`	A comma separated string of values for diagnostic testing. Valid test values are: `LINKCHECK`: Run boot test to check I/O node cabling. The I/O nodes must be cabled. This is the default. `NOLINKCHECK`: Do not run boot test to check I/O node cabling.
`--msglevel`	Controls the output message level. `Verbose`: Detailed output messages. `Debug`: Includes all of the output from Verbose
`--dbproperties` *`<path>`*	The fully qualified file name for the db.properties file. The default is /bgsys/local/etc/db.properties.
`--base` *`<path>`*	The base (installation) path to the files that are used by this command. The default is /bgsys/drivers/ppcfloor.
`--log` *`<path>`*	The path to the directory that contains the log file from this command. The default is /bgsys/logs/BGP.
`--help`	Displays the help text for the command.

The following example shows the syntax to prepare R00-M0-N4 for service from the command line:

```
./ServiceNodeCard R00-M0-N04 Prepare
```

11.2.2 Additional commands

Two additional command line only commands are available in V1R3M0. Both commands are located in the /bgsys/drivers/ppcfloor/bareMetal directory:

- The **`RecoverServiceAction`** command restores the database after a failed or interrupted Service Action. The command uses the following syntax:

  ```
  RecoverServiceAction <ServiceAction ID>|All [options]
  ```

The <*ServiceAction ID*> is the ID number of the Service Action to close. Specifying `All` closes all open Service Actions. Options for the command are the same as those listed for the commands in 11.2.1, “Command line Service Actions” on page 201.

- The **`VerifyCables`** command verifies the data cable connections for a single link card, all of the link cards in a midplane, rack, or the entire machine. The command uses the following syntax:

  `VerifyCables <LocationString>|ALL [options]`

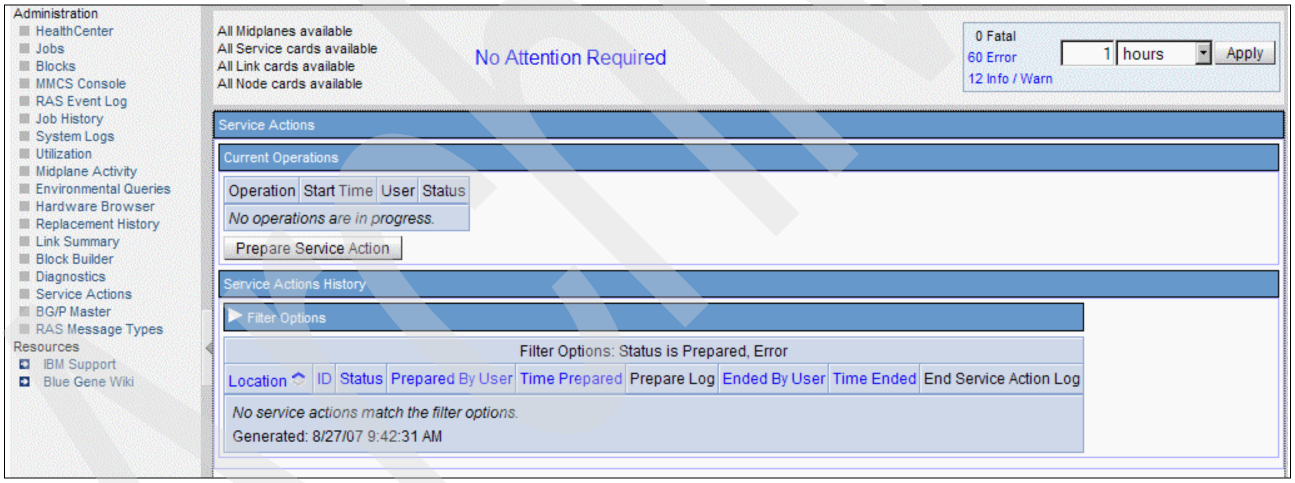
  The possible location strings are:

`Rxx-Mx-Lx`	A specific link card.
`Rxx-Mx`	All of the link cards in the specified midplane.
`Rxx`	All of the link cards in the specified rack.
`ALL`	All of the cables in the machine.

  The **`VerifyCables`** command has one additional option, `--reset`, that the other commands do not have. Specifying `--reset YES` resets the link chips in the specified location. The default for the command is `--reset NO`, which does *not* reset the link chips.

11.2.3 Using Service Actions from the Navigator

In Blue Gene Navigator, you click the Service Action link in the navigation pane to open the page shown in Figure 11-1. In addition to initiating Service Actions from this page, you can also filter your view to show Service Actions, historical, and current. You can also query by time, location, or user.



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

No Attention Required

Current Operations:

Operation	Start Time	User	Status
No operations are in progress.			

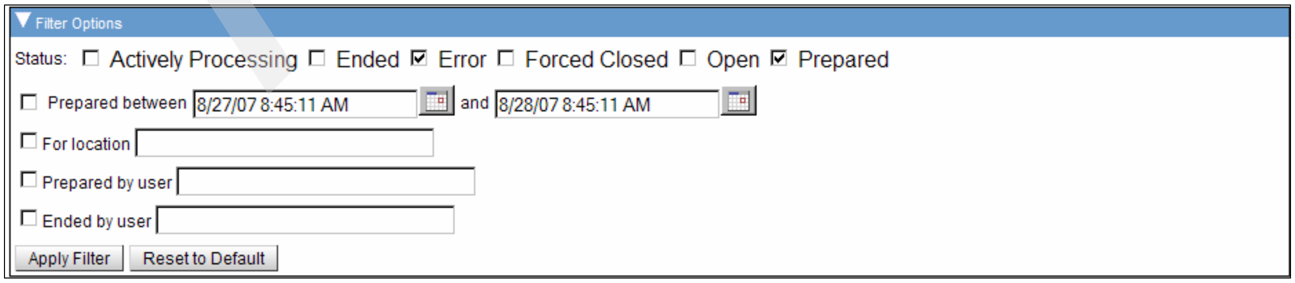
Filter Options: Status is Prepared, Error

Location	ID	Status	Prepared By User	Time Prepared	Prepare Log	Ended By User	Time Ended	End Service Action Log
No service actions match the filter options.								

Generated: 8/27/07 9:42:31 AM

Figure 11-1 Service Actions link in Navigator

Figure 11-2 shows the possible filters that you can apply to query the BGPSERVICEACTION table.



Status: ☐ Actively Processing ☐ Ended ☑ Error ☐ Forced Closed ☐ Open ☑ Prepared

☐ Prepared between 8/27/07 8:45:11 AM and 8/28/07 8:45:11 AM

☐ For location

☐ Prepared by user

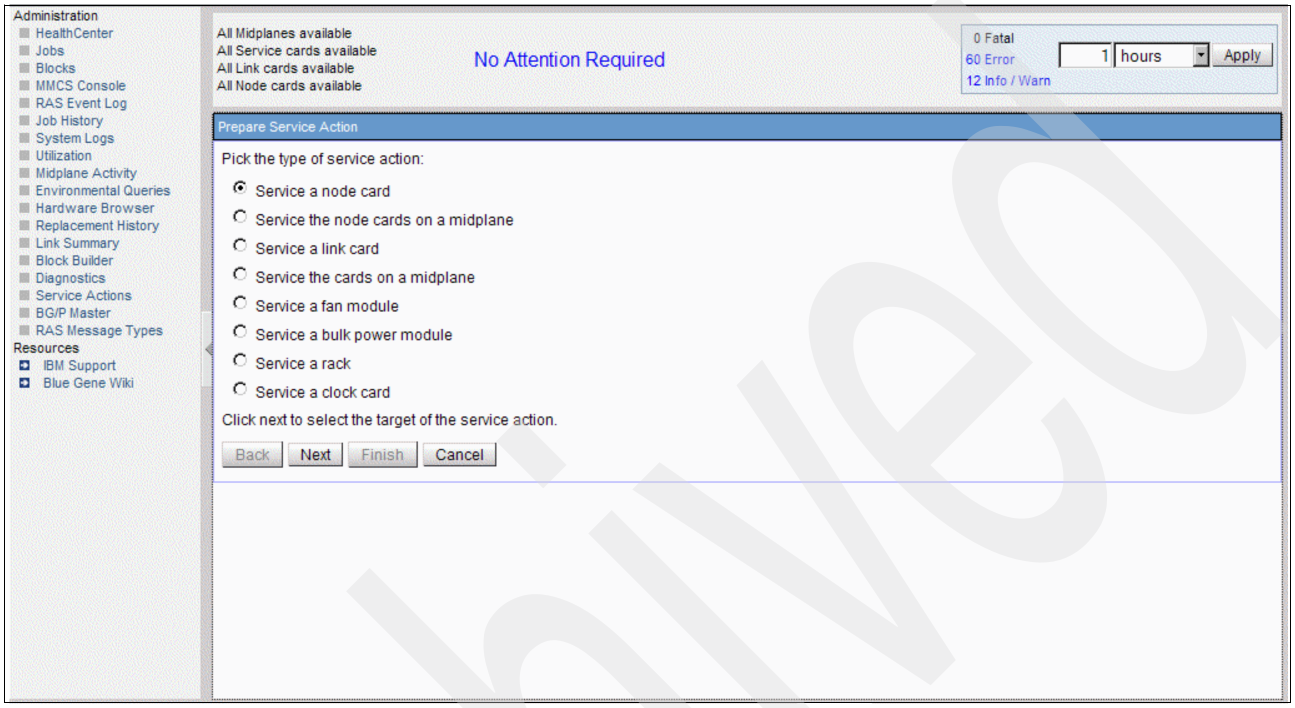
☐ Ended by user

Figure 11-2 Service Action filters

Starting a Service Action

To start a Service Action, follow these steps:

1. Click **Prepare Service Action** in the content area of the page to open the page shown in Figure 11-3. From this window, you can choose the hardware that you want to service. Click **Next**.



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

No Attention Required

1 hours

Prepare Service Action

Pick the type of service action:

- [✓ Service a node card]
- Service the node cards on a midplane
- Service a link card
- Service the cards on a midplane
- Service a fan module
- Service a bulk power module
- Service a rack
- Service a clock card

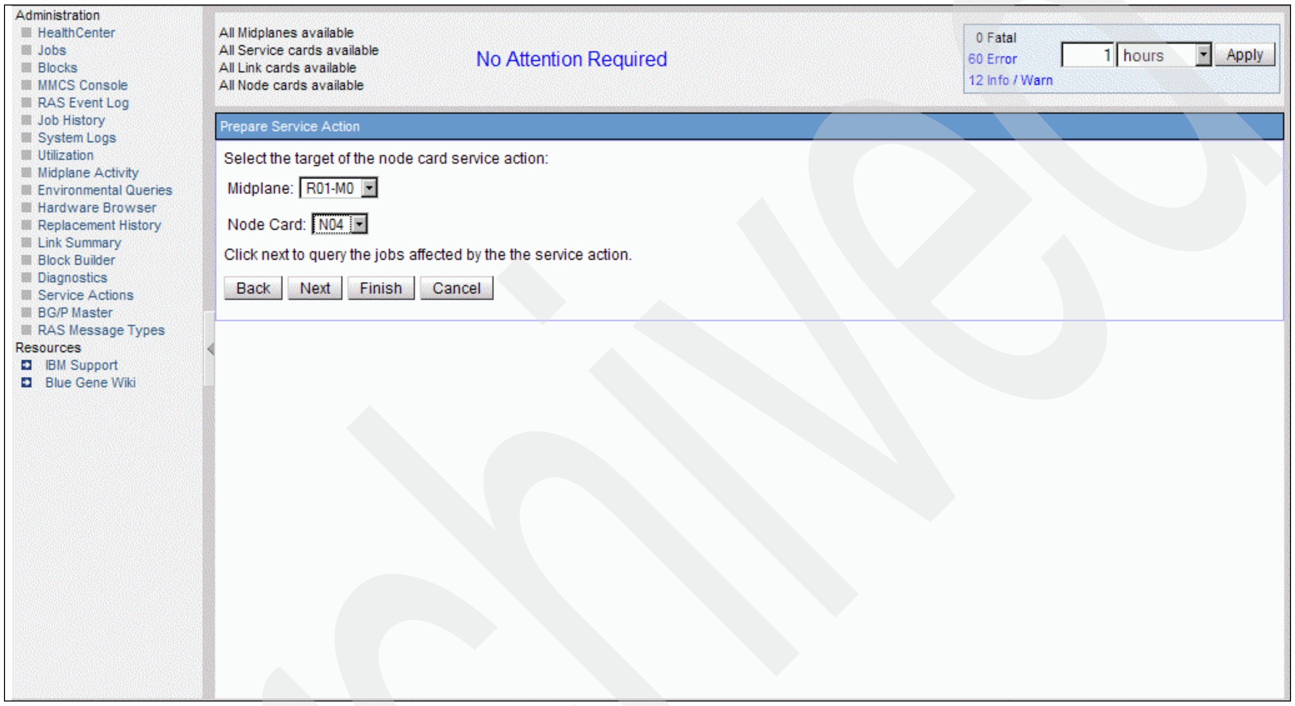
Click next to select the target of the service action.

Figure 11-3 Start a Service Action

2. Select a target location for the Service Action (Figure 11-4). Recall that starting a Service Action on hardware that has a block booted or job running causes that block to be freed or the job to be terminated. Click **Next** to verify that the hardware that you have selected is not in conflict with any running jobs.

   Note: If you are performing a Service Action on a fan module or a bulk power module, the Next button is disabled. These types of Service Actions do not affect jobs.

   If you are not concerned about potential conflicts, you can click **Finish** from this page and proceed with the Service Action.



All Midplanes available
All Service cards available
All Link cards available
All Node cards available

No Attention Required

1 hours

Prepare Service Action

Select the target of the node card service action:

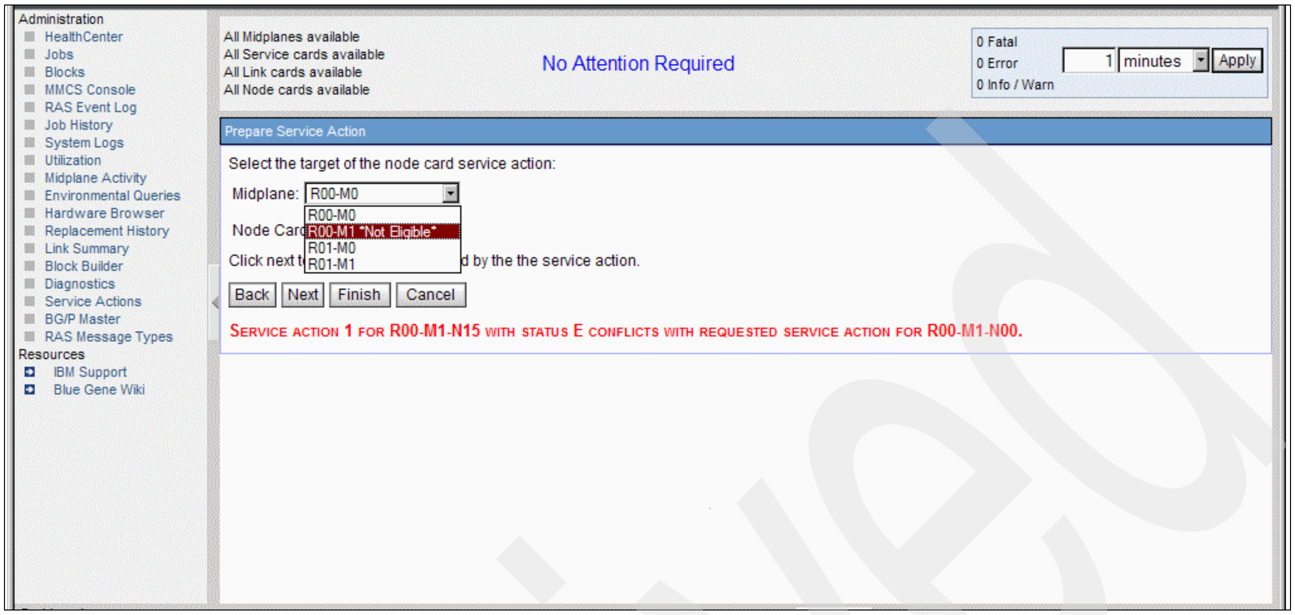
Midplane: R01-M0

Node Card: N04

Click next to query the jobs affected by the the service action.

Figure 11-4 Select the target

If you attempt to start a Service Action for a specific component for which a pending Service Action already exists, you receive a message as shown in Figure 11-5.



Select the target of the node card service action:

Midplane: R00-M0

R00-M0
R00-M1 *Not Eligible*
R01-M0
R01-M1

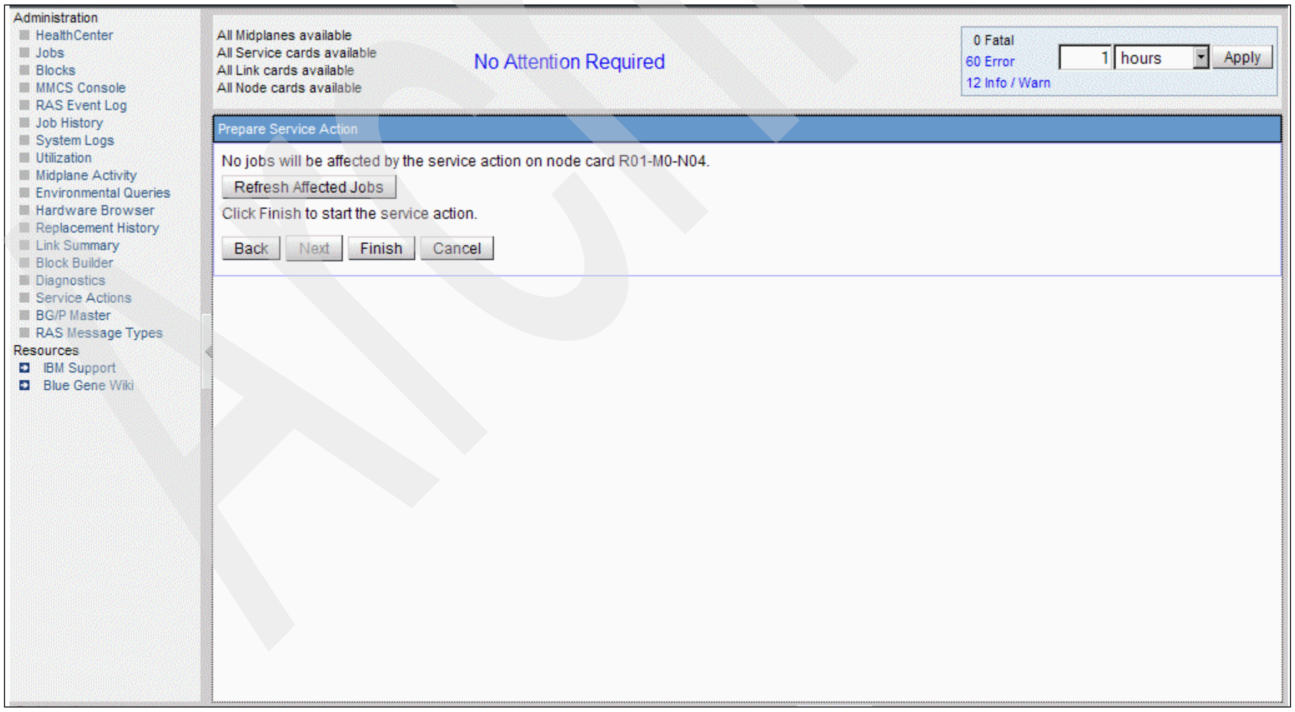
Node Card

Click next t... d by the the service action.

SERVICE ACTION 1 FOR R00-M1-N15 WITH STATUS E CONFLICTS WITH REQUESTED SERVICE ACTION FOR R00-M1-N00.

Figure 11-5 Conflicting Service Actions

3. Messages in the drop-down menu and within the window also alert you to possible conflicts with existing Service Actions. When you are satisfied with your choices, you can click **Finish** to start the Service Action (Figure 11-6).



No jobs will be affected by the service action on node card R01-M0-N04.

Click Finish to start the service action.

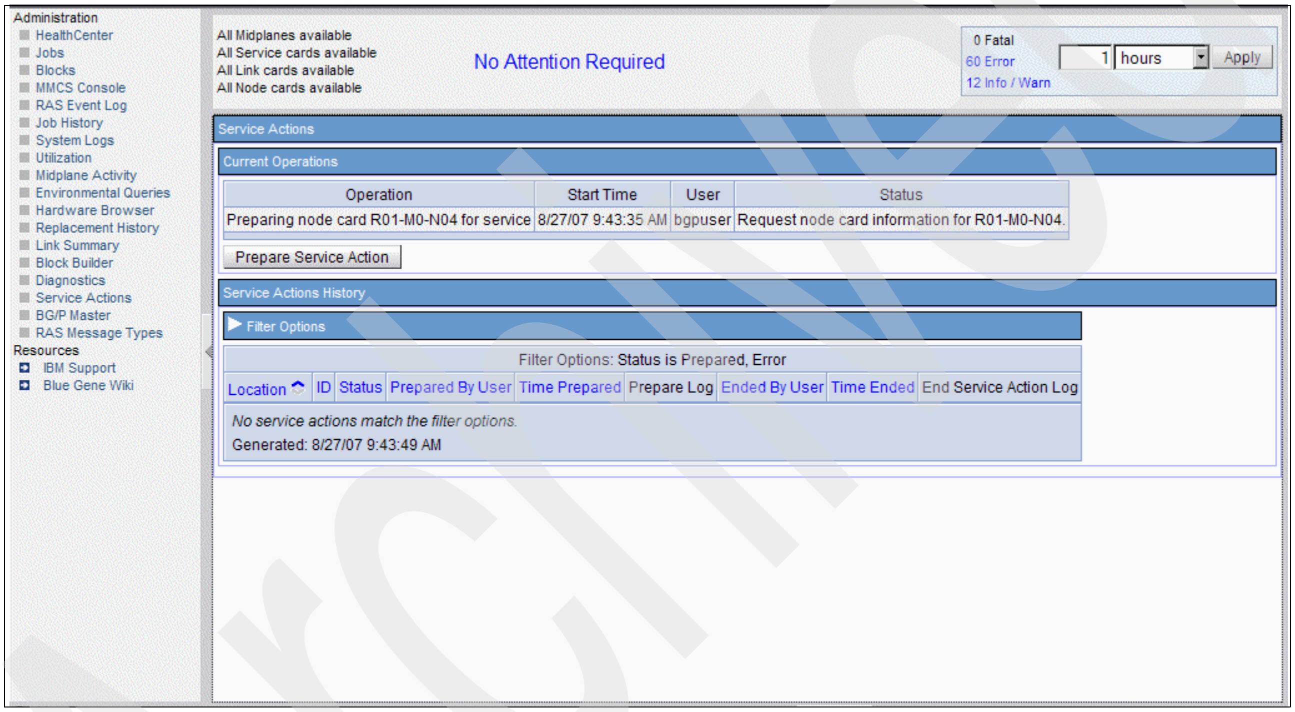
Figure 11-6 Check for conflicts

4. Verify that you want to start the Service Action by clicking **OK** to proceed.
5. Click **Refresh** at the bottom of the window to see the status change as the Service Action progresses (Figure 11-7).

Tip: In most cases, you need to use the Refresh button at the bottom of the Navigator window. Because of the type of data that is contained in the pages (POSTDATA), using the browser's refresh feature can cause errors.

When running a Service Action on a node card, you see the following status changes:

– Request node card information
– Starting
– Issuing write VPD (the card becomes unavailable at this point)
– Issuing power off request for node card Rxx-My-Nzz



Operation	Start Time	User	Status
Preparing node card R01-M0-N04 for service	8/27/07 9:43:35 AM	bgpuser	Request node card information for R01-M0-N04.

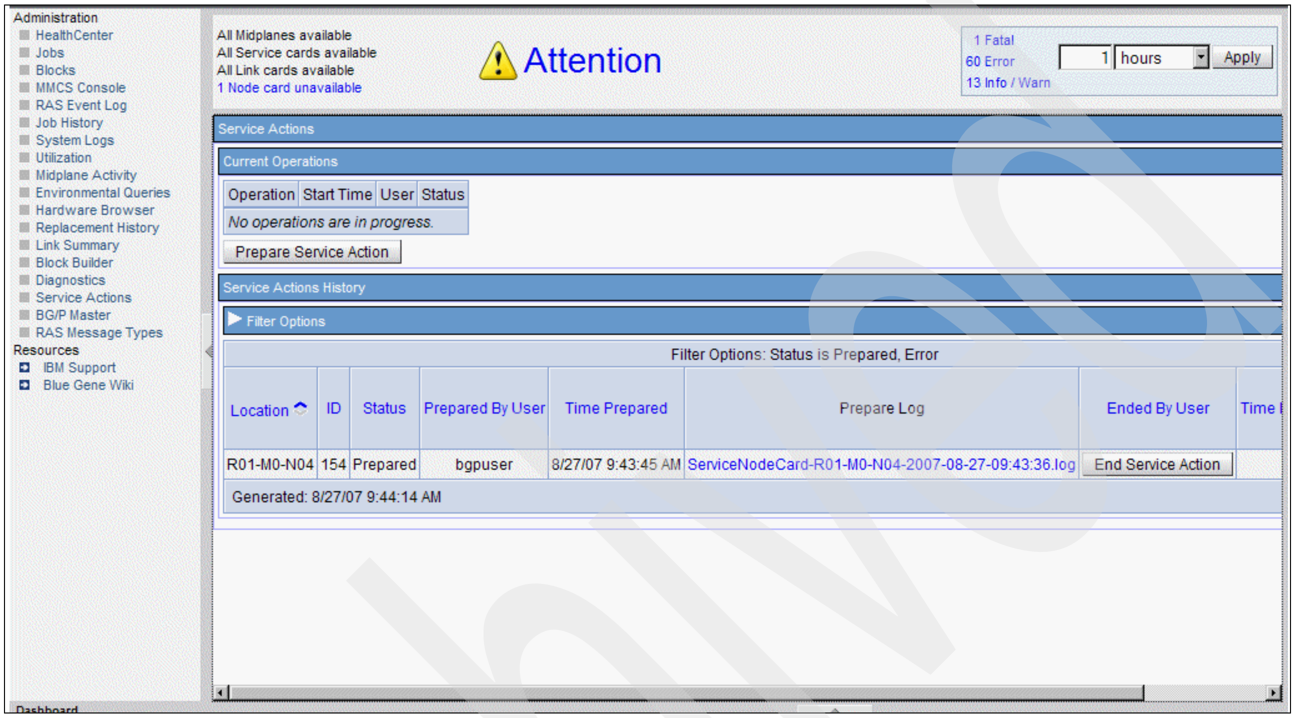
Filter Options: Status is Prepared, Error

Location	ID	Status	Prepared By User	Time Prepared	Prepare Log	Ended By User	Time Ended	End Service Action Log

No service actions match the filter options.
Generated: 8/27/07 9:43:49 AM

Figure 11-7 Initial Service Action page

6. When the specified hardware is ready for service, the Service Action window changes as shown in Figure 11-8. Note the status, in this case *Prepared*, which indicates that the prepare Service Action was successful. It is important to verify the status prior to performing any service on the system. For example, if the status is *Error*, the power might not have terminated properly and removing hardware at this point can be hazardous not only to the hardware itself, but also to the service personnel.



All Midplanes available
All Service cards available
All Link cards available
1 Node card unavailable

Operation	Start Time	User	Status
No operations are in progress.			

Filter Options: Status is Prepared, Error

Location	ID	Status	Prepared By User	Time Prepared	Prepare Log	Ended By User	Time E
R01-M0-N04	154	Prepared	bgpuser	8/27/07 9:43:45 AM	ServiceNodeCard-R01-M0-N04-2007-08-27-09:43:36.log	End Service Action	

Generated: 8/27/07 9:44:14 AM

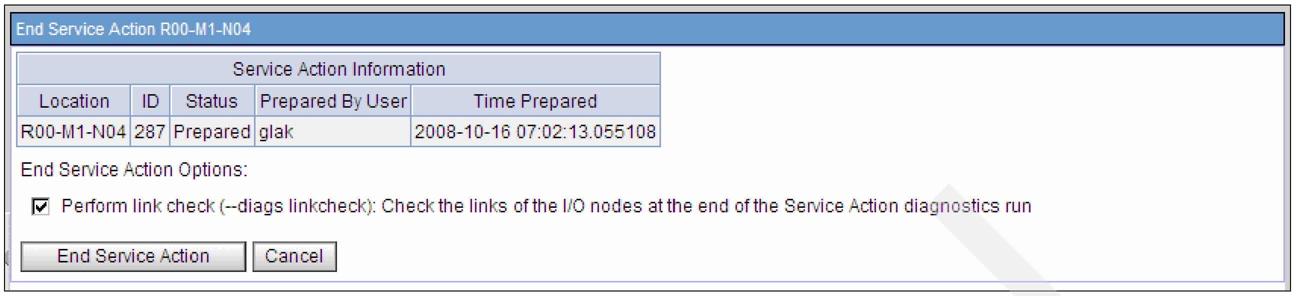
Figure 11-8 Service Action ready

Ending a Service Action

While the Service Action is open, you see the page shown in Figure 11-8. To return the hardware to a usable state, click the corresponding **End Service Action** button for the Service Action that you want to complete. In V1R2M0 and prior releases, you are prompted to verify the action. Click **OK** to proceed.

After clicking the End Service Action button in V1R3M0, a new page (shown in Figure 11-9) opens that allows you to either cancel or confirm the choice to end service. Additionally, depending on the type of Service Action that you are completing, you might be offered the choice to perform a link check. The “Perform link check option” is available on the following Service Actions:

- Node card
- Node cards on a midplane
- Cards on a midplane
- Rack
- Clock card


End Service Action R00-M1-N04

Service Action Information				
Location	ID	Status	Prepared By User	Time Prepared
R00-M1-N04	287	Prepared	glak	2008-10-16 07:02:13.055108

End Service Action Options:

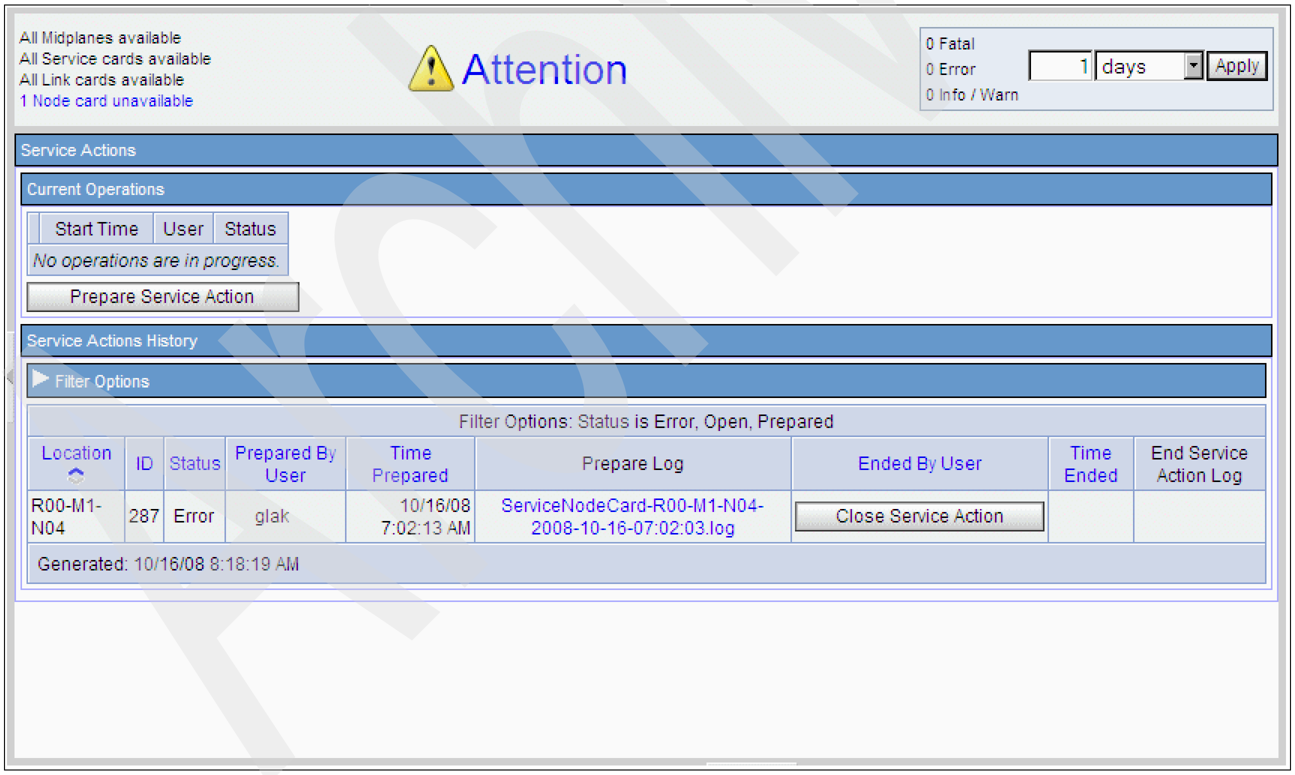
☑ Perform link check (--diags linkcheck): Check the links of the I/O nodes at the end of the Service Action diagnostics run

Figure 11-9 Link check

In the Current Operations section of the main content area, you see the Status column make the following changes as the hardware is re-initialized:

1. Issues initialize node card request.
2. Requests node card information.
3. Initializes node card Rxx-My-Nzz request completed. (The hardware becomes available at this point.)

Starting in V1R3M0, you can close a Service Action that is in an error state from Navigator. When the status is Error, as shown in Figure 11-10, a Close Service Action button becomes available in the *Ended By User* column. After you select **Close Service Action**, you can confirm that you want to close the Service Action.


All Midplanes available
All Service cards available
All Link cards available
1 Node card unavailable

Attention

0 Fatal
0 Error
0 Info / Warn

Start Time	User	Status
No operations are in progress.		

Filter Options: Status is Error, Open, Prepared

Location	ID	Status	Prepared By User	Time Prepared	Prepare Log	Ended By User	Time Ended	End Service Action Log
R00-M1-N04	287	Error	glak	10/16/08 7:02:13 AM	ServiceNodeCard-R00-M1-N04-2008-10-16-07:02:03.log	Close Service Action		

Generated: 10/16/08 8:18:19 AM

Figure 11-10 Service Action in Error

11.3 Cycling power

The process to cycle power on your system depends on the reason that you are shutting down the system and on the number of Blue Gene/P racks in your system. Any time that you replace parts within an individual rack, you need to run a Service Action to prepare the hardware for service. When the Service Action ends, the database updates to reflect the new hardware that was installed. If you are shutting down the system with no intention of replacing hardware or with the intention to replace hardware in multiple racks, you can use the alternative process that we describe in this section.

Important: When ending a Service Action, the system is surveyed to find any new hardware. Replacing hardware without a Service Action causes the database to be out of sync with the hardware that is actually installed in the system.

11.3.1 Single rack system

Prior to turning off the power on a rack, you need to prepare the rack properly. Preparing a rack to turn off the power requires that you run the **`ServiceRack`** command. The steps to power off a single rack are:

1. First prepare the rack for service from either the Navigator or the command line. Use the following command line syntax to start a service action on a rack:

   ```
   $ ServiceRack Rxx PREPARE
   ```

   To prepare a rack for service from the Navigator interface, follow the steps listed in 11.2.3, "Using Service Actions from the Navigator" on page 203. In the first step, select the *Service a rack* option (see Figure 11-3 on page 204).

2. The **`ServiceRack`** process terminates any jobs that are running on that specific rack and powers down all the hardware in the rack with the exception of the clock card. After the **`ServiceRack`** command completes, you receive a message that indicates that the rack is ready for service. At this time you can turn off power to the rack.

   If you do not turn off the rack, you must reseat one of the Bulk Power Modules (BPM) to bring the power back on line before ending the Service Action. Reseating a BPM provides the master service card with power which then turns on the remaining Bulk Power Modules.

3. End the Service Action from the command line using the following command:

   ```
   $ ServiceRack Rxx END
   ```

   To end the Service Action from Blue Gene Navigator, click the *End Service Action* button that corresponds to your current action.

11.3.2 Multiple rack system

For multiple rack systems, we discuss two possible scenarios in this section:

- The first scenario involves turning off a system that has no hardware to replace.
- The second scenario involves turning off a system for a hardware replacement.

In either scenario, you can use the **`ServiceClockCard`** command on the rack that contains the master clock. The advantage of this method is that all of the jobs on the system are killed and, when the Service Action ends, all of the link cables are verified. Keep in mind, however, that ending this type of service action verifies only new hardware on the rack that contains the master clock.

Option 1: Turning off a system that has no hardware to replace

In this scenario, we choose not to use the **ServiceClockCard** command. The steps to turn off a multiple rack system when there is no hardware to replace are as follows:

1. Stop the BGP Master process using the following command:

   ```
   $ ./bgpmaster stop
   ```

2. Turn off all of the racks.

To bring the system back online, use the following steps:

1. Turn on the racks starting with the rack that contains the master clock followed by the secondaries, tertiaries, and then the remaining subordinate racks.
2. Start the BGP Master processes with the following command:

   ```
   $ ./bgpmaster start
   ```

Option 2: Turning off a system for a hardware replacement

The steps for this scenario are similar to the first option, but to discover new hardware you run an Install Service Action (ISA) after the system is restarted. Follow these steps:

1. Stop the BGP Master process using the following command:

   ```
   $ ./bgpmaster stop
   ```

2. Turn off all of the racks.

After the hardware is replaced and you are ready to bring the system back online, follow these steps:

1. Turn on the racks starting with the rack that contains the master clock, followed by the secondaries, tertiaries, and then the remaining subordinate racks.
2. Start the BGP Master processes with the following command:

   ```
   $ ./bgpmaster start
   ```

   The MMCS Server should not be running during the ISA. Run the following command to stop it:

   ```
   $ ./bgpmaster stop mmcs_server
   ```

3. From the /bgsys/drivers/ppcfloor/baremetal directory start the ISA:

   ```
   $ ./InstallServiceAction
   ```

4. If you replaced a Link card, verify that the cabling is correct. From the /bgsys/drivers/ppcfloor/baremetal directory, use the following command:

   ```
   $ ./VerifyCables
   ```

5. Restart the MMCS server:

   ```
   $ ./bgpmaster start mmcs_server
   ```

12

Configuring I/O nodes

This chapter provides information about how to configure I/O nodes for your environment.

Note: Configuring GPFS in the Blue Gene/P environment is beyond the scope of this book. For step-by-step instructions, refer to General Parallel File System Version 3 Release 2 HOWTO for the IBM System Blue Gene/P Solution, which provides Blue Gene-specific information for installing and configuring GPFS. To view the latest version, go to the IBM Publications Center at the following address, select your country, and search on publication number SC23-5230:

`http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi`

12.1 Using the I/O node startup and shutdown scripts

The I/O node configuration centers around startup and shutdown, which is always done in a predefined sequence. To understand how to configure the I/O nodes, you must know which scripts and files are used and how and when they interact.

12.1.1 BusyBox

BusyBox provides the shell for the I/O node. It is a small specialized shell that enables a subset of commands and command options. The I/O node boot scripts run the BusyBox commands. For more information about BusyBox, refer to the following Web address:

http://www.busybox.net/

12.1.2 Site customization directory and the ramdisk

The ramdisk contains the root file system that is needed by the I/O node kernel. Table 12-1 lists the directories and files that are important for I/O nodes.

Table 12-1 Ramdisk contents

Directory or file	Description
`/bgsys`	Network File System (NFS) mount point of /bgsys
`/bin`	NFS mount point of `/bgsys/drivers/ppcfloor/linux/OS/bin`
`/bin.rd`	BusyBox and Blue Gene programs
`/ciod.rd`	Internal working directory for CIOD
`/etc/ntp.conf`	NTPD configuration file
`/etc/rc.status`	Common functions for rc scripts
`/etc/sysctl.conf`	The `sysctl` configuration file for customized configuration parameters
`/etc/init.d/boot.sysctl`	The `sysctl` script that reads configuration file and sets parameters
`/etc/sysconfig/ciod`	CIOD configuration file
`/etc/sysconfig/gpfs`	GPFS configuration file
`/etc/sysconfig/sysctl`	Default `sysctl` parameters
`/etc/syslog.conf`	Syslogd configuration file
`/jobs`	Contains subdirectories named with the job ID of the job or jobs that are running currently in the partition. Each subdirectory contains information about the job.
`/lib`	Shared objects from Blue Gene/P toolchain and symlinks to shared objects in `/bgsys/drivers/ppcfloor/linux/OS/lib`
`/proc/personality`	Binary personality file
`/proc/personality.sh`	Script with personality info that can be sourced by shell scripts
`/sbin`	NFS mount point of `/bgsys/drivers/ppcfloor/linux/OS/sbin`
`/sbin.rd`	BusyBox and Blue Gene programs
`/usr`	NFS mount point of `/bgsys/drivers/ppcfloor/linux/OS/usr`

12.1.3 Startup flow

The I/O node kernel boots at first from the ramdisk and then switches to a combination of ramdisk content and NFS-mounted content later in the boot. The kernel transfers control to `/sbin/init` after the kernel completes initialization. A simple inittab file is provided in the ramdisk that gives control to `/etc/init.d/rc.sysinit`.

The `script rc.sysinit` performs the following operations:

1. Mount the / and `/proc` file systems.
2. Source `/proc/personality.sh` to set variables to personality values.
3. Source `/etc/sysconfig/sysinit` for site-specific I/O node initialization settings.
4. Call `rc.network` to initialize the network, which includes the following actions:
   - Validate the MAC address.
   - Initialize the functional network interface (`eth0`).
   - Initialize the local network interface (`lo`).
   - Add route to network gateway.
5. Mount `/bgsys`, 32 nodes at a time:
   - Copy all files and directories from `/bgsys/iofs` to `/`.
   - Rename `/bin` to `/bin.rd` and make a symbolic link from `/bin` to `/bgsys/drivers/ppcfloor/linux/OS/bin`.
   - Rename `/sbin` to `/sbin.rd` and make a symbolic link from `/sbin` to `/bgsys/drivers/ppcfloor/linux/OS/sbin`.
   - Make a symbolic link from `/usr` to `/bgsys/drivers/ppcfloor/linux/OS/usr`.
   - Make symbolic links from `/lib` to any shared libraries in `/bgsys/drivers/ppcfloor/linux/OS/lib` that do not already exist in `/lib`.

   Note You can configure the path `/bgsys/drivers/ppcfloor/linux/OS/bin` to point elsewhere. You can also set the path using the environment variable `$BG_OSDIR`.

6. Set the host name from `/etc/hosts` if it exists, otherwise use the node location name from the personality.
7. Run **`ldconfig`** to create the shared library cache.
8. Run any scripts of the form `/etc/init.d/rc3.d/S[0-9][0-9]*`.
9. Signal the control system that the I/O node has completed boot.

12.1.4 Node Concurrency

By default all nodes boot simultaneously, which has the potential to create significant load on external resources such as file servers, time servers, and so forth. To help limit this impact, a script called atomic coordinates the activities of the nodes in a block. This script takes as arguments a concurrency factor and the name of a program to run. The specified program is run concurrently only on the specified number nodes.

For example, a concurrency factor of 16 means that only 16 nodes at a time run the program. All other nodes wait for those 16 nodes to finish the specified program. In addition, nodes that finish the program wait for all other nodes to finish the program before continuing. This capability is useful when mounting or unmounting file systems.

As a convenience, the atomic script is used when running scripts that are named according to the following convention:

```
[S | K][0-9][0-9]*.[1-9] | [1-9][0-9] | [1-9][0-9][0-9]
```

If your site script performs NFS mounts, it might be useful to limit the concurrency of the mount requests by naming the script `S10sitefs.32`. In a similar fashion, if your site script unmounts remote file systems, it might be named `K10sitefs.32`. For these examples, only 32 nodes at a time run the specified program.

The atomic script uses the Blue Gene/P global barrier network to synchronize the nodes so the cost to wait for nodes to finish the program is on the order of a few microseconds. Note that the script assumes that all nodes are using atomic to run the specified program. If even one node fails to call atomic, all the other nodes hang waiting for the missing node or nodes to call atomic.

12.1.5 Shutdown flow

When an I/O node halts. the shutdown rule in `/etc/inittab` runs. This rule executes the script `/etc/rc.shutdown`. The shutdown flow is basically the reverse of the startup flow, although it is less complicated. The shutdown flow is as follows:

1. Run any scripts of the form `/etc/init.d/rc3.d/K[0-9][0-9]*`.
2. Flush the file system buffers.
3. Unmount all file systems.
4. Bring down all network interfaces.
5. Call the BusyBox implementation of halt.

12.1.6 Shell variables from personality info

The special file `/proc/personality.sh` is a shell script that sets shell variables to node-specific values. See Table 12-2. These variables are used by the `rc.*` scripts and might also be useful within site-written start and shutdown scripts. They are not exported variables, so each script must invoke this file with the **`source`** command or export the variables to other scripts.

Table 12-2 Shell variables available during I/O node startup and shutdown

Shell variable	Description
BG_BGSYS_FS_TYPE	Specifies which NFS version should be used to mount `/bgsys`. Possible values are: NFSv3 or NFSv4.
BG_BLOCKID	Name of the block
BG_BROADCAST	Broadcast address for this I/O node
BG_CIO_MODE	Mode for running CIO services. Possible values are: ► FULL: All CIO services are run ► MUX_ONLY: Only the CIO multiplexer daemon is run ► NONE: No CIO services are run
BG_CLOCKHZ	Processor frequency in megahertz
BG_EXPORTDIR	File server export directory for `/bgsys`
BG_FS	File server IP address for `/bgsys`
BG_GATEWAY	Gateway address for the default route for this I/O node
BG_GLINTS	Non-zero value when global interrupts are enabled

Shell variable	Description
BG_HTC_MODE	Equal to 1 if HTC mode is enabled, 0 otherwise
BG_IP	IP address of this I/O node
BG_IS_IO_NODE	Equal to 1 if the current node is an IO node, 0 otherwise
BG_ISTORUS	String to indicate if torus HW is present (normally empty string on I/O nodes)
BG_LOCATION	Location string for this I/O node, of the form Rrr-Mm-Nnn-Jjj where rr is the rack number, m is the midplane number, nn is the node number, and jj is the slot number. for example, R04-M1-N01-J01 Use this variable rather than hard coding the I/O node location names.
BG_MAC	Ethernet MAC address of this I/O node
BG_MTU	MTU for this I/O node
BG_NETMASK	Netmask to be used for this I/O node
BG_NODESINPSET	Number of compute nodes served by this I/O node
BG_NUMPSETS	Total number of psets in this block (that is, the number of I/O nodes)
BG_PSETNUM	Processor set (pset) number for this I/O node (`0..$BG_NUMPSETS-1`)
BG_PSETORG	Shows the X, Y, Z coordinates for nodes
BG_PSETSIZE	String with three numbers, size of X dimension, size of Y dimension, and size of Z dimension
BG_RANK_IN_PSET	A node's number within its PSET
BG_SIMULATION	Non-zero if block is running under simulation
BG_SN	Functional network IP address of the service node
BG_UCI	Universal component ID for this I/O node
BG_VERBOSE	Non-zero if the block is created with the i*o_node_verbose* option"
BG_XSIZE, BG_YSIZE, BG_ZSIZE	Size of block in nodes, one variable for each of the three dimensions

12.1.7 I/O node log file

Output written to the console is sent through the mailbox to the control system and stored in a log file in the `/bgsys/logs/BGP` directory. The log file name is the I/O node's location string with `.log` appended to the end (for example, `R01-M0-N04-J01.log`). Output from the kernel, startup and shutdown scripts, and daemons is put in the log file and can be helpful for diagnosing problems or errors.

12.2 Configuring I/O node services

This section addresses the configuration of I/O node services including the Control and I/O Daemon (CIOD), the Network Time Protocol (NTP) daemon, Syslog, and the Network File System (NFS).

12.2.1 Configuring CIO daemon

The Control and I/O Daemon (CIOD) is started by the `/etc/init.d/rc3.d/S50ciod` script. The configuration information for CIOD is stored in the `/etc/sysconfig/ciod` file. Table 12-3 lists the environment variables used by CIOD. You can change the default options for CIOD by providing a `/bgsys/iofs/etc/sysconfig/ciod` file. For example, to change the default read/write buffer size to 512 KB, the `/etc/sysconfig/ciod` file should contain the following line:

```
export CIOD_RDWR_BUFFER_SIZE=524288
```

Environment variables used by CIOD

Table 12-3 describes the environment variables used by CIOD and when they were implemented.

Table 12-3 Environment variables used by CIOD

Environment variable	Description
CIOD_ENABLE_ CONTROL_TRACE	When the value is non-zero, CIOD turns on a control trace when it starts. The default value is 0.
CIOD_END_ JOB_TIMEOUT	The value is the number of seconds CIOD waits before it assumes the ioproxy process is hung when the job is ended. When the timeout expires, CIOD ends the ioproxy process by sending a SIGKILL signal. The default value is 4.
CIOD_END_TOOL_ TIMEOUT	The value is the number of seconds CIOD waits before it assumes the external tool is hung when it is being ended. When the timeout expires, CIOD ends the external tool process by sending a SIGKILL signal. The default value is 10.
CIOD_IOPROXY_ AFFINITY_MASK	The value is a bit mask that controls what cores the ioproxy processes are allowed to run on. For example, to allow the ioproxy processes to run on cores 0, 1, and 2, the value is 7. The default value is 15 to allow the ioproxy processes to run all cores.
CIOD_LOG_ LEVEL	The value controls the amount of data written by CIOD to the I/O node log file. When the value is 0, CIOD does no additional logging. When the value is 1, CIOD logs additional information for recoverable errors. When the value is 2, CIOD also logs when jobs start and end. When the value is 3, CIOD also logs performance data for selected system calls. The log level is accumulative so that when the value is 3 the information for log levels 1 and 2 is also written to the log file. The default value is 0.
CIOD_MAIN_ AFFINITY_MASK	The value is a bit mask that controls what cores the main CIOD threads are allowed to run on. For example, to allow the main threads to run on cores 0 and 1, the value is 3. The default value is 15 to allow the main threads to run all cores.

Environment variable	Description
CIOD_MAX_ CONCURRENT_ LOAD_OPS	The value is the maximum number of concurrent program load operations from this I/O node. A value of 0 means there is no limit to the number of concurrent load operations. Note that with a limit it is possible for blocking operations to keep this I/O node from making progress. The default value is 4 when the block is running in HTC mode. Otherwise the default value is 0.
CIOD_MAX_ CONCURRENT_ RDWR_OPS	The value is the maximum number of concurrent read and write operations from this I/O node. A value of 0 means there is no limit to the number of concurrent operations. Note that with a limit it is possible for blocking operations to keep this I/O node from making progress. The default value is 0.
CIOD_NEW_JOB_ EXIT_PGM	The value is the path to a program that is called each time a new job is started. The program is passed eight parameters describing the new job (see 12.2.2, “CIOD new job program” on page 220 for details). If the program returns a non-zero exit status, CIOD returns an error to the control system and the job does not start. The default value is null.
CIOD_NEW_JOB_ EXIT_PGM_ TIMEOUT	The value is the number of seconds that CIOD waits before it assumes the new job exit program is hung. When the timeout expires, CIOD ends the program, returns an error to the control system and the job does not start. When the value is -1, CIOD waits forever. The default value is -1.
CIOD_READER_ AFFINITY_MASK	The value is a bit mask that controls what core the collective network reader thread is allowed to run on. For example, to allow the reader thread to run on core 3, the value is 8. The default value is 15 to allow the reader thread to run all cores.
CIOD_READER_ INTERRUPT_MODE	When the value is non-zero, the collective network reader thread waits to be interrupted when a packet is available in the collective network reception fifo. When the value is zero, the reader thread polls the collective network reception fifo looking for packets. The default value is 0.
CIOD_RDWR_ BUFFER_SIZE	This value specifies the size, in bytes, of each buffer used by CIOD to issue read and write system calls. One buffer is allocated for each compute thread associated with this I/O node. The default size, if not specified, is 262144 bytes. The minimum size is 8096 bytes. A larger size might improve performance. If you are using a GPFS file server, we recommend that you ensure that this buffer size matches the GPFS block size. Experiment with different sizes, such as 262144 (256 K) or 524288 (512 K), to achieve the best performance.
CIOD_REPLY_ MSG_SIZE	The value is the size in bytes for reply messages sent from CIOD to the compute nodes. The value cannot be larger than the value of the CIOD_RDWR_BUFFER_SIZE variable. A smaller value causes CIOD to return the data from a system call in multiple messages and spend shorter intervals injecting packets into the collective network device. The default value is 262144.
CIOD_TRACE_ DIRECTORY	The value is the path to the directory CIOD uses to store trace files. The default value is NULL.
CIOD_TREE_ RECV_RETRY	The value is the number of times that CIOD tries to receive a packet from the collective network device before it decides that a packet is not available and posts ready messages. A larger value causes CIOD to pay more attention to the collective network. The default value is 50.

12.2.2 CIOD new job program

CIOD can call a program provided by the administrator each time a new job starts. For example, the program can be used to tune the file system client on a per job basis or mount specific file systems based on the user.

The program is specified by setting the CIOD_NEW_JOB_EXIT_PGM environment variable in the `/etc/sysconfig/ciod` file. The program runs as root in a new process group. This allows the program to perform the same functions as an rc script that runs at startup or shutdown. When the program runs, all of the compute nodes have loaded the application and are waiting to start the application.

A timeout can be specified by setting the CIOD_NEW_JOB_EXIT_PGM_TIMEOUT environment variable in the `/etc/sysconfig/ciod` file. If the program does not complete in the number of seconds specified by the timeout, CIOD ends the process group by sending a SIGKILL signal. The default value of -1 causes CIOD to wait forever.

If there is an error in starting the program, CIOD allows the job to continue. If the program returns a non-zero exit status or waiting for the program to complete times out, CIOD sends an error reply back to the control system. The control system then marks the job as *failed* and CIOD resets and waits for the next job.

The program is passed the following parameters:

- Indicator of job mode (*0* for coprocessor mode, *1* for virtual node mode or *2* for dual mode)
- Number of physical compute nodes managed by CIOD on this I/O node
- Number of compute nodes processes that are participating in the job
- User ID of current user
- Group ID of current user
- Job number of the job
- Number of jobs run so far on this boot
- Path to program running on compute nodes

12.2.3 Configuring Network Time Protocol

The Network Time Protocol daemon (NTPD) is started by the `/etc/init.d/rc3.d/S05ntp` script. The configuration information for the NTPD is stored in the `/etc/ntp.conf` file. If the file does not exist, a default file is created by the S05ntp script, which includes the following lines:

```
restrict default nomodify
server $BG_SNIP
```

You can change the default options for NTPD by providing a `/bgsys/iofs/etc/ntp.conf` file. For example, you can change the NTP server to something other than the service node.

You must also set your time zone file. The I/O node looks for the time zone in the file `/etc/localtime`. To set up your time zone file, copy your service node's `/etc/localtime` file to `/bgsys/iofs/etc/localtime`. If the time zone file is not found, the time is assumed to be UTC.

For more information about NTP, refer to the following Web address:

http://www.ntp.org/

12.2.4 Configuring the Syslog daemon

The syslogd and klogd daemons are started by the `/etc/init.d/rc3.d/S02syslog` script. The default system log file name is the I/O node's location string with `.log` appended. For example:

```
R01-M1-N12-J01.log
```

The configuration information for syslogd is stored in the `/etc/syslog.conf` file. Example 12-1 shows the contents of the default file that is included with the ramdisk.

Example 12-1 Contents of syslog.conf

```
# /etc/syslog.conf - Configuration file for syslogd(8)
# For info about the format of this file, see "man syslog.conf".
#
# print most on tty10 and on the xconsole pipe
#
kern.warning;*.err;authpriv.none /dev/tty10
kern.warning;*.err;authpriv.none|/dev/xconsole
*.emerg     *

# enable this, if you want that root is informed
# immediately, e.g. of logins
#*.alert    root
#
# Warnings in one file
#
*.=warning;*.=err-/var/log/warn
*.crit      /var/log/warn
#
# save the rest in one file
#
*.*;mail.none;news.none-/var/log/messages
#
# enable this, if you want to keep all messages
# in one file
#*.*       -/var/log/allmessages
#
# Some foreign boot scripts require local7
#
local0,local1.*-/var/log/localmessages
local2,local3.*-/var/log/localmessages
local4,local5.*-/var/log/localmessages
local6,local7.*-/var/log/localmessages
```

You can change the default options for syslogd and klogd by providing a `/bgsys/iofs/etc/syslog.conf` file. The log files can get quite large. Consider using a utility to manage these files, such as logrotate.

For more information, refer to the man pages for syslogd, klogd, and logrotate.

12.3 I/O node performance tips

You can improve I/O performance on Blue Gene/P with CIOD buffer size and NFS mount options. This section contains some areas where you might try tweaking these areas to enhance I/O node performance.

12.3.1 Configuring the CIOD buffer size

You might improve I/O performance by increasing the size of the CIOD buffers that are used to issue read and write system calls. One CIOD buffer is allocated in the I/O node per compute node thread in the processor set (pset). You can change the buffer size (in bytes) by creating a `/bgsys/iofs/etc/sysconfig/ciod` file to set the CIOD_RDWR_BUFFER_SIZE environment variable. See 12.2.1, “Configuring CIO daemon” on page 218 for details about configuring CIOD.

Note: The buffer size value is per compute node thread. For example, if the CIOD read/write buffer size is set to 256 KB and there are 32 compute nodes per processor set with each compute node having four threads, then CIOD allocates a total of 32768 KB for read/write buffers.

12.3.2 Configuring Network File System

Blue Gene/P currently supports Network File System (NFS) V3 and version V4. To set up NFS for Blue Gene/P on the service node, follow these steps:

1. Configure the number of threads that the NFS server will use in the `/etc/sysconfig/nfs` file. The default for the `USE_KERNEL_NFSD_NUMBER` is four. You can adjust that number to suit your environment (see the next section, 12.3.3, “Performance tips for the NFS file server”).
2. Next, export the file system. Edit the `/etc/exports` file, and add the following line (assuming the I/O nodes will be in the `172.16.x.x` network):

   ```
   /bgsys   172.16.0.0/255.255.0.0
   ```

   Then export using the following command:

   ```
   exportfs -ar
   ```
3. Now you can mount `/bgsys` from a client using the following command:

   ```
   mount -o soft 172.16.1.1:/bgsys /bgsys
   ```

12.3.3 Performance tips for the NFS file server

Consider the following points when you perform the configuration on your NFS file server on the service node:

- The number of NFS server daemons running can be a source of performance issues. If you have too few daemons running, they can become overloaded, which can increase wait times. You can check the use of the **`nfsd`** threads using the following command:

  ```
  # grep th /proc/net/rpc/nfsd
  ```

  The results look similar to the following line:

  ```
  th 8 594 3733.140 83.850 96.660 0.000 73.510 30.560 16.330 2.380 0.000 2.150
  ```

  In this case, there are eight **`nfsd`** threads running, followed by the number of times all of the threads have been needed (594). The 10 numbers that follow show how long, in seconds, a certain fraction of those threads have been busy, starting with less than 10%

and ending with over 90%. If the last numbers are high, you can increase the **nfsd** thread count. Increase the number of NFS server daemons, using the `USE_KERNEL_NFSD_NUMBER=`**"***xxx***"** parameter in the `/etc/sysconfig/nfs` file. Try using 32, 64, or 128.

- If you use the *noac* (no attribute cache) option when mounting NFS directories to avoid problems with multiple users writing to the same file, you might experience slower performance. Administrators should consider mounting different file systems with different attributes to achieve the best performance.
- On a Linux server, use the **sysctl** command to change the settings that pertain to TCP and sockets.

Note: The following values might reflect the extreme. The values you supply will depend on your system and memory.

– This setting turns off TCP time stamp support; the default is on.

```
net.ipv4.tcp_timestamps = 0
```

– This sets the TCP time-wait buckets pool size; the default is 180000.

```
net.ipv4.tcp_max_tw_buckets =2000000
```

– This sets the min/default/max TCP read buffer; the defaults are 4096, 87380, and 174760.

```
net.ipv4.tcp_rmem =10000000 10000000 10000000
```

– This sets the min/pressure/max TCP write buffer; the defaults are 4096, 16384, and 131072.

```
net.ipv4.tcp_wmem =10000000 10000000 10000000
```

– This sets the min/pressure/max TCP buffer space; the defaults are 31744, 32256, and 32768.

```
net.ipv4.tcp_mem =10000000 10000000 10000000
```

– This turns off SACK support; the default is on.

```
net.ipv4.tcp_sack =0
```

– This turns off TCP window scaling support; the default is on.

```
net.ipv4.tcp_window_scaling =0
```

– This is the maximum number of skb-heads to be cached; the default is 128.

```
net.core.hot_list_length =20000
```

– This is the maximum receive socket buffer size; the default is 131071.

```
net.core.rmem_max =10000000
```

– This is the maximum send socket buffer size; the default is 131071.

```
net.core.wmem_max =10000000
```

– This is the default receive socket buffer size; the default is 65535.

```
net.core.rmem_default =10000000
```

– This is the default send socket buffer size; the default is 65535.

```
net.core.wmem_default =10000000
```

– This is the maximum amount of option memory buffers; the default is 10240.

```
net.core.optmem_max =10000000
net.core.netdev_max_backlog =300000
```

- On the NFS mount command, described in Table 12-4 on page 226, specifying larger values for `rsize` and `wsize` might improve performance. Other mount options to consider include:
  - TCP versus UDP mount

    Mounting an NFS share with the UDP protocol on a saturated network can be problematic because you might see a high level of lost packets. Generally, it is best for performance to use TCP only for mounts on a very active network.
  - Asynch versus Sync mount

    If you export file systems in NFS using the `sync` option, you might be adversely affecting general performance. You might be able to improve overall performance on the clients using the default asynchronous writes.

    The `async` option allows the server to reply to the client as soon as it processes the request, so that the client does not have to wait for the disk write to complete. The client continues under the assumption that the write completed.

    There is a slight risk in using the `async` option, but only in the situation of a disk failure. The `sync` option guarantees that all writes are completed when the client thinks they are, but it does so by forcing each client to wait for disk writes to complete.

12.3.4 NFS errors

This section contains a few of the more common NFS errors and how to debug them.

RPC: Connection Refused

The two main RPC programs that NFS uses are **rpc.mountd** and **portmap**. The `RPC: Connection Refused` error (received from the I/O node) indicates that one of these programs might not be running. You can verify that the programs are running with the following commands on the system running the NFS server (typically the service node):

```
ps -ef | grep portmap
ps -ef | grep rpc.mountd, or rpcinfo -p
```

If you find that **portmap** is not running, you can start it using the following command:

```
# /etc/init.d/portmap start
```

If both **rpc.mountd** and **nfsd** are not running, you can start both with the following command:

```
# /etc/init.d/nfsserver start
```

Otherwise, start **rpc.mountd** with the following command:

```
# /usr/sbin/rpc.mountd
```

Also, verify that both programs are configured to start on boot with the following commands:

```
# chkconfig –list portmap
# chkconfig –list nfsserver
```

If the programs are not configured correctly, set them to start automatically with the following commands:

```
# chkconfig –add nfsserver
# chkconfig –add portmap
```

In some situations, **rpc.mountd** might crash as a result of excessive load. In past Linux versions, **rpc.mountd** was a single-threaded user-level process. As of SLES 10 SP2, you can now pass the **-t** flag with **rpc.mountd** to specify the number of **rpc.mountd** threads to spawn. To increase performance and to prevent **rpc.mountd** from crashing due to load, we recommend that you update to the SP2 version of **nfsutils** (nfs-utils-1.0.7-36.29). To take advantage of this option on boot, you need to edit the /etc/init.d/nfsserver file. There are two lines in the nfsserver file that can potentially start **rpc.mountd**. Adding the **-t** flag, as shown in Example 12-2, starts **rpc.mountd** with 16 threads.

Example 12-2 Use of the -t flag

```
...
if [ -n “$MOUNTD_PORT” ] ; then
startproc /usr/sbin/rpc.mountd -t 16 -p $MOUNTD_PORT
else
startproc /usr/sbin/rpc.mountd -t 16
fi
...
```

RPC: Timeout

This error usually occurs on mount requests when **rpc.mountd** is overloaded. If **rpc.mountd** is overwhelmed, these timeouts occur when running commands such as **showmount -e <SN IP Address>** from another Linux system. To lessen the impact of these issues:

- Space out mount requests to **rpc.mountd**.
- Verify DNS host resolution of all involved servers. Add the host name and IP addresses for the front end node, service node, and I/O node to the /etc/hosts file on the service node. Resolving names and addresses by sending requests to the DNS servers occupies the **rpc.mountd** threads for a longer period of time than if the resolution is done locally. Other mount requests are dropped if they are waiting on a DNS reply. Also, the hosts line in the /etc/nsswitch.conf file should be hosts: files dns to give preference to /etc/hosts over the DNS servers.
- Increase the number of **rpc.mountd** threads. As discussed in 12.3.3, “Performance tips for the NFS file server” on page 222, as of SLES 10 SP2, you can now pass **rpc.mountd** the **-t** flag to specify the number of **rpc.mountd** threads to spawn. We suggest updating to the SP2 version of **nfsutils** (nfs-utils-1.0.7-36.29). To take advantage of this option on boot, you need to edit the /etc/init.d/nfsserver script as shown in Example 12-2.

Mount failure: Permission Denied

This error is usually generated due to either a missing entry in /etc/exports, a syntax error in that file, or due to a mismatch between /etc/exports and /var/lib/nfs/etab.

Missing entry in /etc/exports

The format of an /etc/exports entry is as follows:

```
/filesystem hosts(options)
```

A typical entry for exporting /bgsys looks like the entry shown in Example 12-3.

Example 12-3 Export of /bgsys

```
/bgsys 172.16.0.0/255.255.0.0(rw,async,no_root_squash)
/tmp/gpfsvar
134.94.72.0/255.255.254.0(rw,no_root_squash,async,fsid=746)
```

After adding an entry to `/etc/exports`, run **`exportfs -ar`** to make the change known to the NFS daemons. The **`exportfs`** command makes the change known by modifying `/var/lib/nfs/etab`.

Syntax error in /etc/exports

If there is an unknown option or if the line does not conform to the expected format, the **`exportfs`** command does not add a line to `/var/lib/nfs/etab` for the file system. Check for any malformed networks, host names, or file system names in the `/etc/exports` file. Also, it is not possible to export both a directory and its child (for example both `/usr` and `/usr/local`). Export the parent directory with the necessary permissions, and then mount all of its subdirectories with those same permissions. After you make any changes, run the **`exportfs -ar`** command again to make the changes known to the NFS daemons.

Mismatch between /etc/exports and /var/lib/nfs/etab

A `Permission Denied` error on a mount request might simply be the result of someone forgetting to run the **`exportfs`** command after editing `/etc/exports`. The NFS daemons really use the files in `/var/lib/nfs/` as they run. If the file system is in `/etc/exports` but not in `/var/lib/nfs/etab`, then you need to run the **`exportfs -ar`** command to make the changes known to the NFS daemons.

12.4 Typical site customization

A Blue Gene/P site normally has a script that customizes the startup and shutdown. This script can perform any or all of the following functions in the order shown:

1. Mount (during startup) and unmount (during shutdown) high performance file servers for application use. To ensure that the file servers are available to applications:
   - The mount must occur after portmap is started but before CIOD is started by the `S50ciod` script.
   - The unmount must occur after CIOD is ended by the `K50ciod` script and before portmap is ended.

   For NFS mounts, we recommend that you retry the mount several times to accommodate a busy file server.

   See Table 12-4 for the recommended mount options.

Table 12-4 Recommended mount options for NFS

Option	Discussion
tcp	This option provides automatic flow control when it detects that packets are being dropped due to network congestion or server overload.
rsize wsize	These options specify read and write NFS buffer sizes. The minimum recommended size is 8192, and the maximum recommended size is 32768. If your server is not built with a kernel compiled with a 32768 size, it negotiates down to what it can support. In general, the larger the size is, the better the performance is. However, depending on the capacity of your network and server, a size of 32768 might be too large and cause excessive slowdowns or hangs during times of heavy I/O. Each site needs to tune this value.
async	Specifying this option might improve write performance, although there is greater risk for losing data if the file server crashes and is unable to get the data written to disk.

2. Set the CIOD environment variables. See 12.2, “Configuring I/O node services” on page 218.
3. Set NTP parameters. See 12.2, “Configuring I/O node services” on page 218.
4. Set syslog parameters. See 12.2.4, “Configuring the Syslog daemon” on page 221.

The site customization script should be in the `/bgsys/iofs/etc/init.d` directory (for example called `sitefs`). Then, to place it properly in the startup and shutdown sequence, create symbolic links to this script as follows:

```
ln -s /bgsys/iofs/etc/init.d/sitefs /bgsys/iofs/etc/init.d/rc3.d/S10sitefs
ln -s /bgsys/iofs/etc/init.d/sitefs /bgsys/iofs/etc/init.d/rc3.d/K90sitefs
```

The `S10sitefs` and `K90sitefs` links are sequenced so that they meet the mount requirements described previously:

- During startup, `S10sitefs` is called with the *start* parameter.
- During shutdown, `K90sitefs` is called with the *stop* parameter.

Example 12-4 illustrates a site customization script that mounts a user file system at `/bgusr`. An application called `bgras` is used to generate a RAS event if `/bgusr` cannot be mounted.

Example 12-4 Sample site customization script

```
#!/bin/sh
#
# This will run after portmap so NFS can be mounted with locking.
# It runs before CIOD so the mounted filesystems will be available for the
# first job that might run.
#

# Redirect to the console.
exec > /dev/console 2>&1

. /etc/rc.status

SITEFS=172.16.1.101

rc_reset

case "$1" in
        start)
        echo "Mounting site filesystems"
        [ -d /bgusr ] || mkdir /bgusr

        let ATTEMPT=1
        let ATTEMPT_LIMIT=5
        NFS_OPTS=mountvers=3,rsize=8192,wsize=8192,tcp,async,lock,noac
        until test $ATTEMPT -gt $ATTEMPT_LIMIT || mount $SITEFS:/bgusr /bgusr -o
$NFS_OPTS; do
                sleep $ATTEMPT
                let ATTEMPT=$ATTEMPT+1
        done
        if test $ATTEMPT -gt $ATTEMPT_LIMIT; then
                echo "All attempts to mount $SITEFS:/bgusr failed... giving up."
                bgras 1 34 2 "Mount of $SITEFS:/bgusr failed."
        fi
        rc_status -v
        ;;

        stop)
        echo "Unmounting site filesystems"
        umount /bgusr -l

        rc_status -v
        ;;

        restart)
        echo "Restart not implemented."
        ;;

        status)
        echo "Status not implemented."
        ;;
esac
rc_exit
```

A more advanced script, such as the one shown in Example 12-5, can select a file server based on the I/O node location string, for example R01-M0-N12-J02. Note that `BG_LOCATION` comes from invoking `/proc/personality.sh` with the **`source`** command.

Example 12-5 Selecting file server based on I/O node location string

```
case "$BG_LOCATION" in
   R00-*) SITEFS=172.32.1.1;;# rack 0 NFS server
   R01-*) SITEFS=172.32.1.2;;# rack 1 NFS server
   R02-*) SITEFS=172.32.1.3;;# rack 2 NFS server
   R03-*) SITEFS=172.32.1.4;;# rack 3 NFS server
   R04-*) SITEFS=172.32.1.5;;# rack 4 NFS server
   R05-*) SITEFS=172.32.1.6;;# rack 5 NFS server
esac
```

The script can use $BG_IP (the IP address of the I/O node) to compute the file server or servers to select. It can also use $BG_BLOCKID so that different blocks can mount special file servers.

12.4.1 Customizing and building the ramdisk

Although building a custom ramdisk is relatively simple, doing so is usually unnecessary. All files and directories under `/bgsys/iofs` are copied to the ramdisk during boot, thus overriding any conflicting files in the process. You might find it necessary to build a custom ramdisk if you need to make modifications to the boot process before `/bgsys/iofs` is accessible or if you need to remove some files present in the default ramdisk.

If customization or rebuilding of the ramdisk is necessary, follow these steps:

1. To rebuild the default ramdisk, run the following command from the `/bgsys/drivers/ppcfloor/boot` directory:

   ```
   ../build-ramdisk
   ```

   This command creates a backup copy of the current ramdisk image if it conflicts with the name of the new ramdisk and generates a new ramdisk image in `/bgsys/drivers/ppcfloor/boot`. You can change the name of the generated ramdisk if desired using **`-o`** option.

   To see all options, use the following command:

   ```
   build-ramdisk --help
   ```

2. To add a subdirectory to the root of the ramdisk use the following commands:

   ```
   cd /bgsys/drivers/ppcfloor/boot
   ../build-ramdisk --addtree <sub directory>
   ```

   These commands create a backup copy of the current ramdisk image if it conflicts with the name of the new ramdisk and generates a new ramdisk image in `/bgsys/drivers/ppcfloor/boot`, which contains the specified subdirectory.

Example 12-6 shows a sample of modifying the existing ramdisk.

Example 12-6 Modifying existing ramdisk

```
> cd /tmp
> mkdir -p myRamdisk/rootfs
> cd myRamdisk
> cp -r /bgsys/drivers/ppcfloor/ramdisk/* rootfs

# make desired modifications to existing files under rootfs directory

> /bgsys/drivers/ppcfloor/build-ramdisk --rootfs rootfs --verbose
build-ramdisk: root file system: rootfs
build-ramdisk: runos: /bgsys/linux/1.4.020090511
build-ramdisk: init:
build-ramdisk: final dirs in ramdisk:
        /bgsys
        /bin
        /ciod.rd
        /dev
        /etc
        /etc/init.d
        /etc/init.d/rc3.d
        /etc/pam.d
        /etc/security
        /etc/sysconfig
        /jobs
        /lib
        /opt
        /proc
        /sbin
        /sys
        /tmp
        /usr
        /usr/bin
        /usr/lib
```

```
        /usr/sbin
        /var
        /var/empty
        /var/lib
        /var/lib/empty
        /var/lib/nfs
        /var/lib/nfs/rpc_pipefs
        /var/lib/nfs/sm
        /var/lib/nfs/sm.bak
        /var/lock
        /var/lock/subsys
        /var/log
        /var/run
build-ramdisk: writing /tmp/ramdisk.gencpio.script.20587 for gen_init_cpio
build-ramdisk: compress ramdisk.cpio
build-ramdisk: build ramdisk from ramdisk.cpio.gz
> cp ramdisk /bgsys/drivers/ppcfloor/boot
```

12.4.2 Additional GPFS notes

The GPFS file system is not enabled by default. To enable GPFS daemons and the co-requisite SSH daemon, you must first create the files /bgsys/iofs/etc/sysconfig/gpfs and /bgsys/iofs/etc/sysconfig/ssh. In the /bgsys/iofs/etc/sysconfig/gpfs file, set the variable GPFS_STARTUP to 1. In the /bgsys/iofs/etc/sysconfig/ssh file, declare the variable SSHD_OPTS. Unless you need specific SSH options, you can set SSHD_OPTS to an empty string. For more information about GPFS, refer to the HOWTO mentioned in the note on the first page of this chapter.

If you have customized your startup and shutdown routines, make sure that you have included the following section to insure that GPFS and SSH daemons start:

```
echo "GPFS_STARTUP=1" >> /etc/sysconfig/gpfs
        if [ ! -r /etc/sysconfig/ssh ]; then
                echo "SSHD_OPTS=\"\"" > /etc/sysconfig/ssh
        fi
```

To verify that **sshd** is running on the I/O nodes, follow these steps:

1. From the MMCS console, allocate a block.
2. Using the {i} locate command (from the MMCS console) determine the locations of your I/O nodes.
3. Verify that the daemon is running by using the **write_con** command:

   ```
   {i}write_con /etc/init.d/sshd status
   ```

   If the daemon is running all is OK.

If **sshd** is not running on the I/O nodes, then complete the following steps:

1. Verify that the /bgsys/iofs/etc/ssh/sshd_config file exists. If not you can copy the file from the /etc/ssh directory. The permissions should be set to 755.
2. Verify the following lines in the sshd_config file:

   ```
   # UsePAM yes - #(Ensure this line is commented out)
   PermitRootLogin yes
   ```

3. Verify that you have the /bgsys/iofs/etc/sysconfig directory. Permissions should be 755.

4. Verify that you have the file /bgsys/iofs/etc/sysconfig/ssh. If it does not exist, then you can copy it from the /bgsys/linux/*xxxxxxx*/etc/sysconfig directory.
5. Use the **write_con** command on the I/O node to start the **sshd** (or reboot the I/O node, it should start automatically):

```
{i}write_con /etc/init.d/sshd start
```

Try to SSH to the I/O node, either by host name or IP address. If you still cannot connect check the log files for errors.

If you are going to allow users to log into the I/O nodes, then it is a good idea to copy /etc/passwd, /etc/group, and /etc/shadow into the /bgsys/iofs/etc directory. That way the users can log in to the I/O nodes as themselves.

13

Power management

The Blue Gene/P racks can observe power consumption that is used by applications using an automatic hardware process on each rack. The compute node kernel also makes use of Blue Gene/P real-time power consumption data to control power consumption of the job without affecting application scalability.

This chapter describes the power management features of Blue Gene/P software.

13.1 Blue Gene/P power management

Power management allows management of power that is used while applications are running. It also provides a repeatable set of performance results during power managed application runs. In addition, this approach allows user-level power utilization control of the system.

Power management has two layers:

- *Kernel level* power management uses available power consumption data that is provided by the node card’s DC/DC power converters to indicate that a management event is required.
- *DC/DC converter type* management polls status registers in the DC/DC voltage converters at regular intervals. Consecutive readings in excess of the maximum settings will shut down power to the rack as both a safety mechanism and to protect the hardware.

13.2 Invoking power management

You can invoke power management *reactively* or *proactively*. Both methods achieve the same goal in that they reduce the power consumption by the application. The advantage to using proactive power management is that you can better control the power that is used during the entire application run.

13.2.1 Reactive power management

Reactive power management is initiated when the power consumption data from the DC/DC converters exceeds a predefined limit. If the amount of current exceeds the reactive power management threshold, both the performance and dynamic power consumption is reduced by 10%, and the application continues to run. After initiation, reactive power management continues until the job is complete. The next job on the system does not start under reactive power management, unless that job also exceeds reactive power management thresholds.

13.2.2 Proactive power management

If you plan on running a job that you know can draw too much power, you can invoke proactive power management when submitting the job. Proactive power management is enabled on a per job basis. It uses the same mechanism as the reactive power management, but the user has control over the magnitude of the power and performance trade-offs. The settings invoked for the job end when the job terminates. If the job is performing proactive power management and the reactive power thresholds are tripped, then the job overrides the proactive settings with the reactive power management settings.

Currently, you can append the following environment variables to a **submit_job** to invoke proactive power management:

`BG_POWERMGMTDUR`	Number of microseconds to spend in idle loop
`BG_POWERMGMTPERIOD`	Proactive throttling period (in microseconds)
`BG_POWERMGMTDDRRATIO`	DDR throttling ratio percentage

The following example submits a job (from MMCS console) invoking power management:

```
submit_job <job> <outfile> <mode> BG_POWERMGMTDUR=100 BG_POWERMGMTPERIOD=1000
BG_POWERMGMTDDRRATIO=10
```

In this example, every 1000 microseconds all nodes in the system take an interrupt, sit in a while loop for 100 microseconds and then return (100/1000=10%).

The `POWERMGMTDUR` and `POWERMGMTPERIOD` must be used together. `POWERMGMTDDRRATIO` can be combined with the other two or can be used alone.

13.3 RAS messages

When power management occurs, either reactive or proactive, there are RAS messages generated. When the threshold has been exceeded all nodes in the block enter power management mode.

- Reactive power management causes the following message:

  ```
  KERN_180D Severity: Warning Message Text: Environment Monitor. Partition now in
  reactive power management mode.
  ```

  This message is only emitted from the node at torus location 0,0,0.

- Each node card that exceeds the max power threshold generates the following message:

  ```
  KERN_1809 Severity: Warning Message Text: Environment Monitor threshold warning
  (Max Power). Status=0x00002000 PTMON Warnings=0x0x00000000, Maximum
  Temperature=26 C, 1.2v Domain power=415 watts, 1.8v Domain power=265 watts,
  TotalPower=670 watts EnvError=0x00000000.
  ```

  This message is only emitted from the node at the node card's J04 slot (or potentially from J06 if this is a sub-node card partition).

- If proactive power management is invoked, you see the following message:

  ```
  KERN_180E Severity: Warning Message Text: Environment Monitor. Partition now in
  proactive power management mode.
  ```

  This message is emitted only from the node at torus location 0,0,0.

In addition to the RAS messages that are generated, you can see current jobs that are running with power management invoked in the Navigator. On the Health Center page, these jobs are listed under one of two categories:

- Running Jobs with Proactive Power Management
- Running Jobs with Reactive Power Management

A

Statement of completion

IBM considers installation to be complete when the following activities are complete:

- The Blue Gene/P rack or racks are physically placed in position.
- The cabling is complete, including power, Ethernet, and torus cables.
- The power to the Blue Gene/P racks can be turned on.
- All hardware is displayed in the Navigator and is available.

B

Blue Gene/P hardware naming convention

This appendix provides an overview of how the Blue Gene/P hardware locations are assigned. This naming convention is used consistently throughout both the hardware and software.

Hardware naming conventions used in Blue Gene/P

Figure B-1 shows the conventions used when assigning locations to all hardware except the various cards in a Blue Gene/P system.

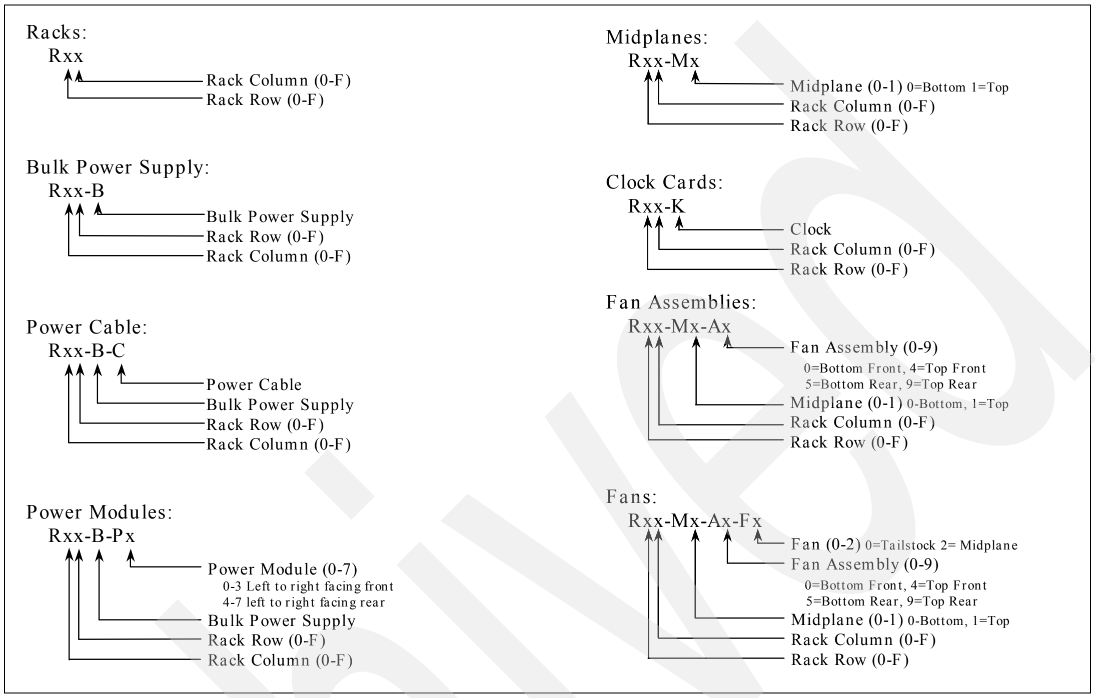


Figure B-1 Hardware naming conventions

As an example, if you had an error in fan R23-M1-A3-0 where would you go to look for it? The upper left corner of Figure B-1 shows racks that use the convention Rxx. Looking at the error message, the rack involved is R23. From the chart, that rack is the fourth rack in row two (remember that all numbering starts with 0). The bottom midplane of any rack is 0, so you are dealing with the top midplane (R23-M1). The chart shows that in the Fan Assemblies description, assemblies zero through four are on the front of the rack, bottom to top, respectively. So, you are going to be checking for an attention light (Amber LED) on the Fan Assembly second from the top, because the front most fan is causing the error message to surface. Service, Link, and Node cards use a similar form of addressing.

Figure B-2 shows the conventions used for the various card locations.

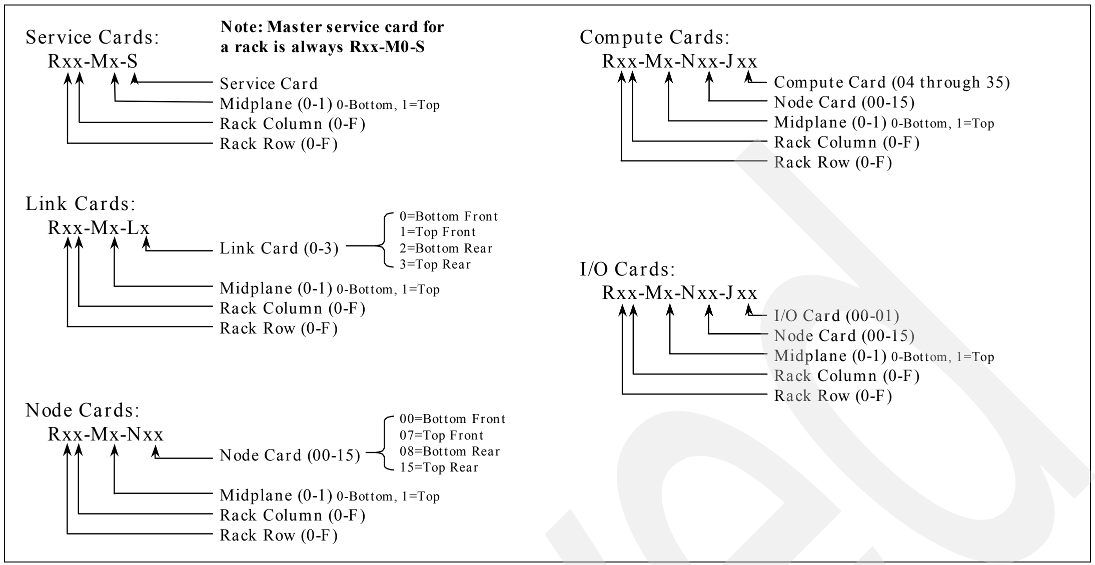


Figure B-2 Card naming conventions

Table B-1 contains examples of various hardware conventions. The figures that follow the table better illustrate the actual hardware.

Table B-1 Example of names

Card	Element	Name	Example
Compute	Card	J04 through J35	R23-M10-N02-J09
I/O	Card	J00 through J01	R57-M1-N04-J00
I/O & Compute	Module	U00	R23-M0-N13-J08-U00
Link	Module	U00 through U05 (00 left most, 05 right most)	R32-M0-L2_U03
Link	Port	TA through TF	R01-M0-L1-U02-TC
Link data cable	Connector	J00 through J15 (as labeled on link card)	R21-M1-L2-J13
Node Ethernet	Connector	EN0, EN1	R16-M1-N14-EN1
Service	Connector	Control FPGA, Control Network *, Clock R, Clock B	R05-M0-S-Control FPGA
Clock	Connector	Input, Output 0 through Output 9	R13-K- Output 3

Figure B-3 shows the layout of a 64 rack system.

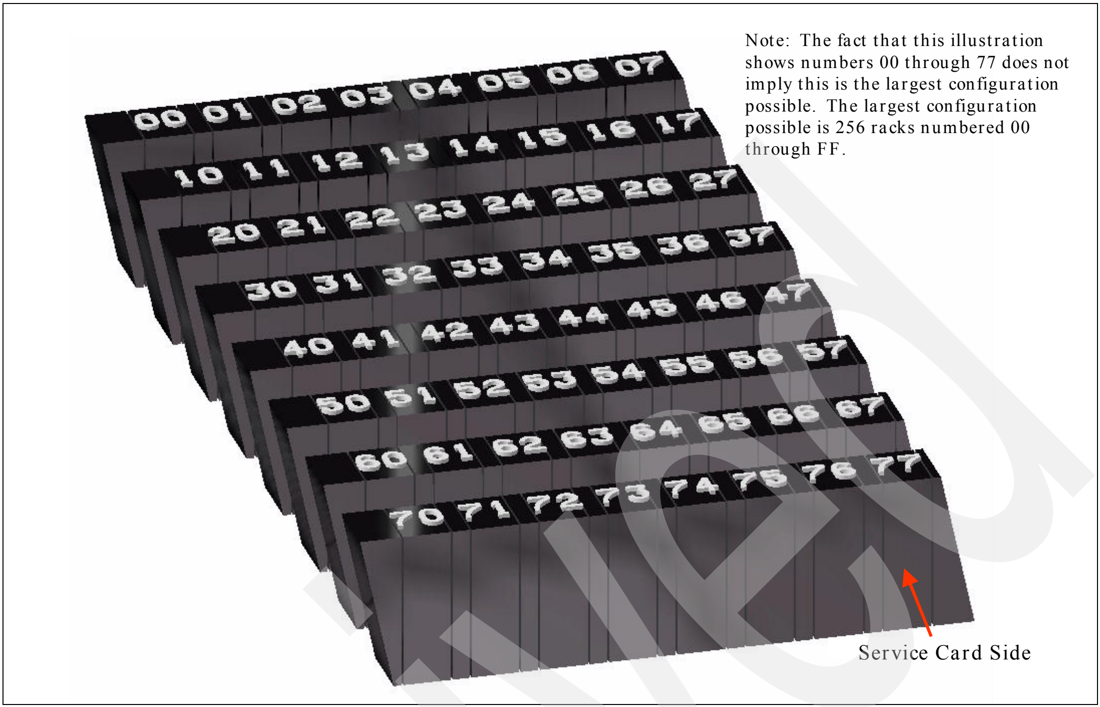


Figure B-3 Rack numbering

Figure B-4 identifies each of the cards in a single midplane.

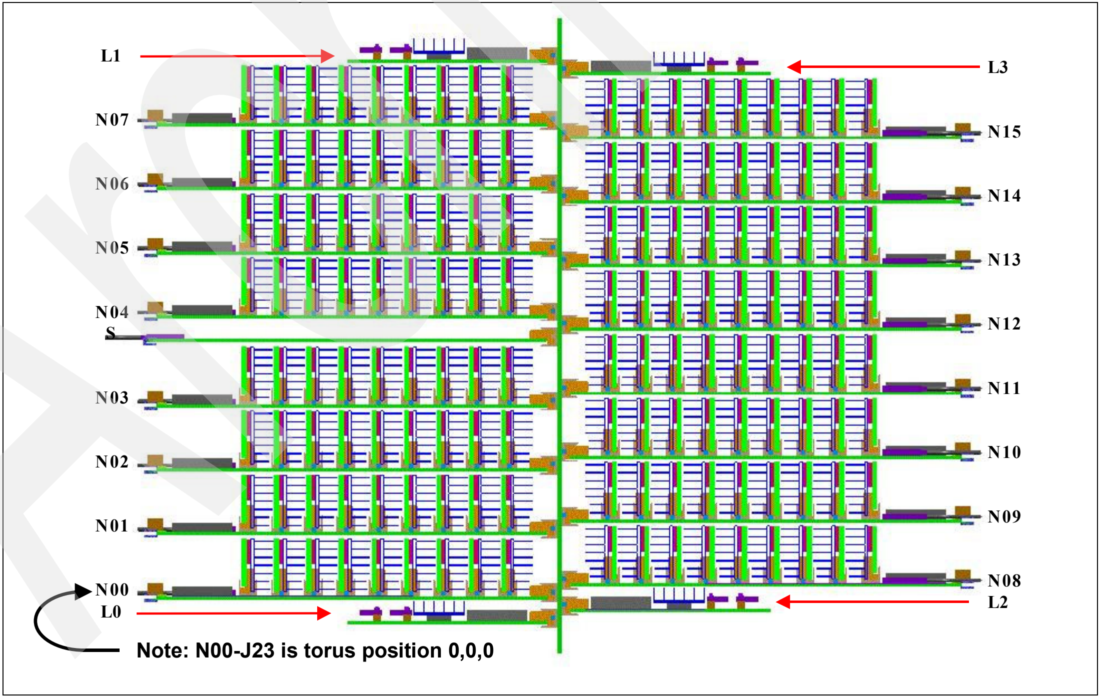


Figure B-4 Positions of Node, Link and Service cards

Figure B-5 is a diagram of a node card. On the front of the card are the Ethernet ports EN0 and EN1. The first to nodes behind the Ethernet ports are the I/O nodes. In this diagram the node card is fully populated with I/O nodes, meaning that it has two I/O nodes. Behind the I/O nodes are the compute nodes.

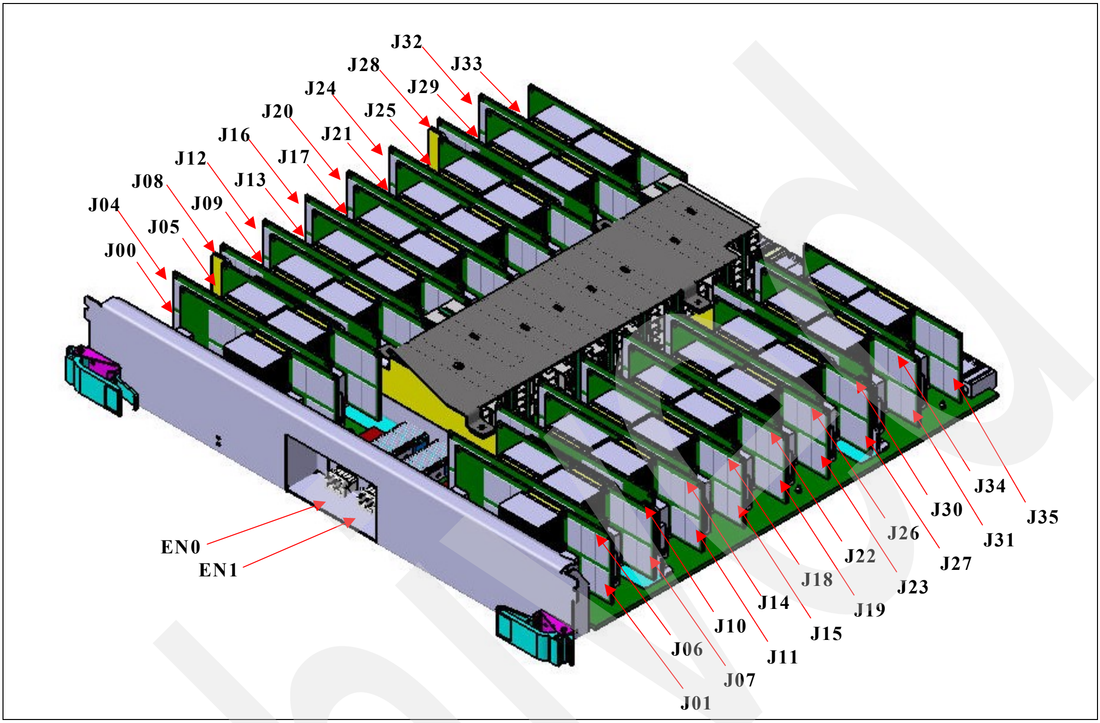


Figure B-5 Node card diagram

Figure B-6 is an illustration of a Service card.

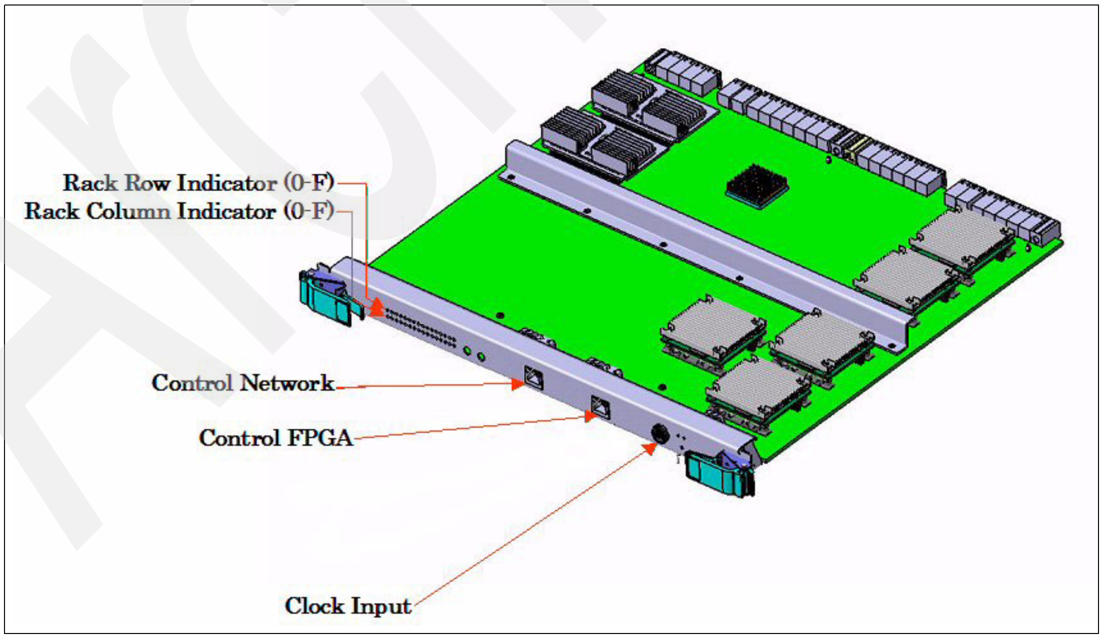


Figure B-6 Service card

Figure B-7 shows the Link card. The locations identified as J00 through J15 are the link card connectors. The Link cables are routed from one link card to another to form the torus network between midplanes.

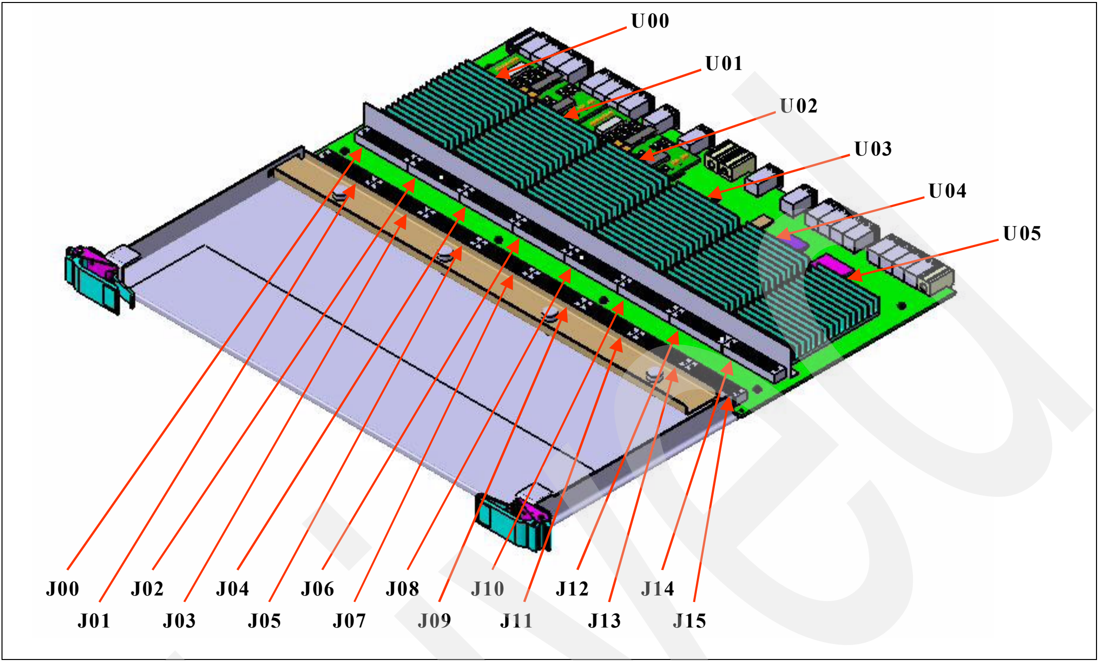


Figure B-7 Link card

Figure B-8 shows the clock card. If the clock is a secondary or tertiary clock there will be a cable coming to the input connector on the far right. Next to the input (just to the left) is the Master/Slave toggle switch. All clock cards are built with the capability of filling either role. If the clock is a secondary or tertiary clock, this must be set to *slave*. Output zero through nine can be used to send signals to midplanes throughout the system.

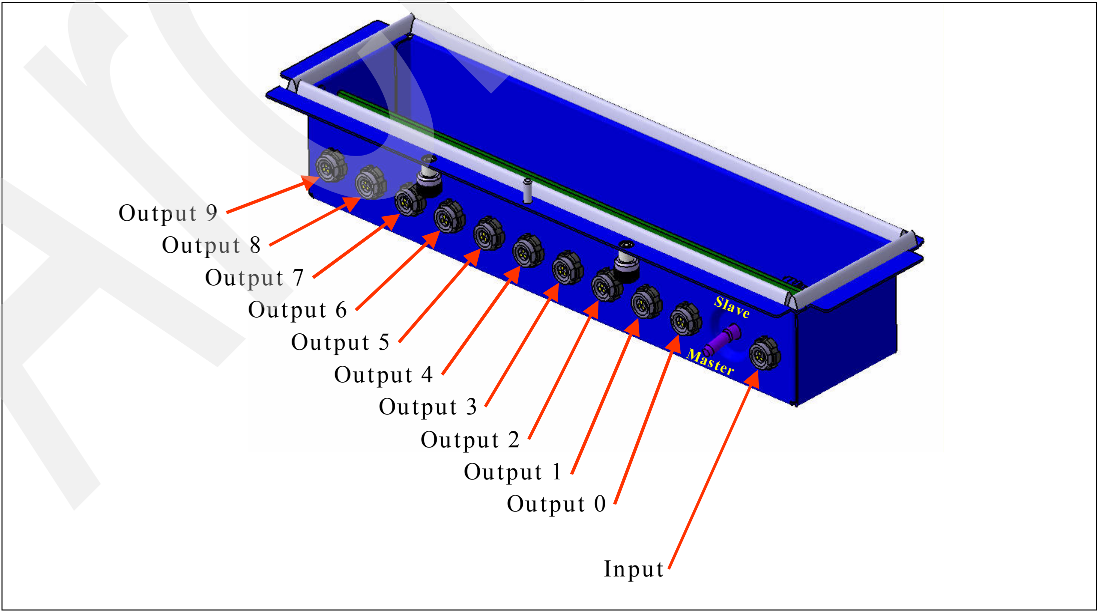


Figure B-8 Clock card

C

Service connection

CIOD supports a service connection that allows the administrator to check on a running CIOD to get status or to enable and disable a trace facility. The administrator enters commands to get an overall status of CIOD and the compute nodes that it is managing. In addition, the administrator can enable and disable a trace facility dynamically that causes CIOD to log control flow.

Accessing the service connection

To access the service connection, you can use two methods:

You can telnet to the I/O node from the service node. To use telnet you must know the TCP/IP address of the I/O node. To determine the TCP/IP address of the I/O node, you can issue the **{i} locate** command on the MMCS console, which returns the addresses of all the I/O nodes in the currently selected block. By default CIOD listens on port 9000 for the service connection. When running in high throughput mode, there is one CIOD for each compute node process. CIOD listens on port 9000 plus the target number of the compute node process. The CIOD multiplexer listens on port 9000 plus the pset size.

Example C-1 shows a sample of a telnet session being initiated where the TCP/IP address of the I/O node is 172.16.60.152.

Example C-1 Telnet session

```
~> telnet 172.16.60.152 9000
Trying 172.16.60.152...
Connected to 172.16.60.152.
Escape character is '^]'.
ciod running with 128 threads and 0 active processes

>
```

Note: The service connection uses the > character as the command prompt.

You can also use the MMCS console **service_ciod** command to access the service connection. The **service_ciod** command allows you to run commands on I/O nodes in the selected block. The output returned by the I/O nodes is sent to the console. The command uses the following syntax:

```
mmcs$ service_ciod [blockID] [command] <options>
```

Service connection commands

The following sections describe the commands that are supported by the CIOD service connection.

Note: In the examples for each of the following commands, the output shows data from a system with 32 compute nodes per IO node. The output is different on a system with a different compute node to IO node ratio.

help

The **help** command displays the syntax of supported commands. Example C-2 shows the output from the help command.

Example C-2 Output from help

```
Commands:
  show_status
  show_states
  show_node_info [nodenumber]
  show_process_info
  show_collective_info
  show_collective_addrs
  show_messages [all|ready|posted|taken|partial|previous]
  show_msgdescriptor_info descriptornumber
  show_version
  show_job_info
  show_connection_info
  show_tool_info
  show_config_info
  show_sync_info
  show_trace_info
  tracedir directory
  trace category on|off
    CONTROL
    DATA
    DEBUGGER
    FILEIO
   INFO
    TEST
  loglevel [level]
  quit
```

quit

The **quit** command ends the service connection session. Example C-3 shows a connection ending.

Example C-3 Output from quit

```
> quit

Happy SuperComputing!
Connection closed by foreign host.
```

show_status

The **show_status** command shows the current high level status of CIOD and overview information about the current job. Example C-4 shows the output of the show_status command.

Example C-4 Output of show_status

```
> show_status
Process id: 752
High throughput: false
Job mode: smp
Job number: 1
Job id: 445345
Program name: /bgusr/sam/IOR
Block name: R00-M0-N08
Torus dimensions: X=4, Y=4, Z=2, T=1
Number of threads: 128
Number of processes: 32
Number of running processes: 32
Number of participating processes: 32
Number of control messages: 128
Number of debugger messages: 0
Number of stdin messages: 0
Number of stdout messages: 0
Next control stream message: END_MESSAGE (8)
Target number: 0

Main thread status:             WAIT_FOR_MSG
Control message thread status:  WAIT_FOR_MSG
Stdin message thread status:    WAIT_FOR_MSG
Stdout message thread status:   WAIT_FOR_MSG
Debugger message thread status: WAIT_FOR_MSG
Debugger thread status:         STOPPED
Service thread status:          CONNECTED
```

Table C-1 provides an explanation of the each of the fields returned by the show_status command.

Table C-1 Description of show_status output

Field	Description
Process id	The process ID of the CIOD process.
High throughput	True if HTC mode is active.
Job mode	The mode of the last job started on the block. The values are smp, dual, or virtual node.
Job number	The Job number field is 0 when CIOD starts, and increments each time a job is started.
Job id	The job identifier from the control system for the last job started on the block.
Program name	The path to the last program started on the block.
Block name	The name of the block the IO node is a part of.
Torus dimensions	Size of the X, Y, Z, and T dimensions for the block.
Number of threads	The number of compute node threads that CIOD is managing.
Number of processes	Total number of compute node processes in the current job.
Number of running processes	The number of compute node processes currently running a program for the current job.
Number of participating processes	The number of compute node processes participating in the current job.
Number of control messages	The total number of control protocol messages processed since CIOD started.
Number of debugger messages	Total number of debugger protocol messages processed since CIOD started.
Number of stdin messages	Total number of stdin protocol messages processed since CIOD started.
Number of stdout messages	Total number of stdout protocol messages processed since CIOD started.

Field	Description
Next control stream message	The type of the next message expected on the control steam from the control system.
Target number	Identifies the compute node process in the pset when running in high throughput mode.
Main thread status	The thread status fields indicate what each thread in the CIOD process is currently doing.
Control message thread status	
Stdin message thread status	
Stdout message thread status	
Debugger thread status	
Service thread status	

show_states

The **`show_states`** command shows the number of compute node threads in each state. The valid states are:

`UNKNOWN`	The compute node threads are in UNKNOWN state when CIOD starts.
`WAITING_TO_LOAD`	The compute node threads is ready to load an application.
`WAITING_TO_START`	The compute node thread has successfully loaded the application and is ready to start.
`RUNNING`	The compute node thread is running the application,.
`ENDING`	CIOD has sent a terminating signal to the compute node thread.
`FINISHED`	The compute node thread is done running the application.
`WAITING_TO_RESET`	The compute node thread is waiting for the other threads in the job to end.

Example C-5 gives provides a sample of the output from the **`show_states`** command.

Example C-5 Output from show_states

```
> show_states
Number of threads UNKNOWN:           0
Number of threads WAITING_TO_LOAD:  128
Number of threads WAITING_TO_START:   0
Number of threads RUNNING:           0
Number of threads ENDING:            0
Number of threads FINISHED:          0
Number of threads WAITING_TO_RESET:   0
```

show_node_info

The **`show_node_info`** command uses the following syntax:

```
show_node_info [nodenumber]
```

The `nodenumber` option represents the number of the compute node thread.

The **`show_node_info`** command shows either summary information for all compute node threads or detailed information for a specified compute node. When the `nodenumber` operand is not specified each line of the output shows summary information about a compute node thread as shown in Example C-6.

Example: C-6 Output from show_node_info with no operand

```
> show_node_info
Index 000:  state=RUNNING         rank=000007, location=R00-M0-N08-J30-C00, X-Y-Z-T=03-01-00-0
Index 001:  state=RUNNING         rank=000003, location=R00-M0-N08-J31-C00, X-Y-Z-T=03-00-00-0
Index 002:  state=RUNNING         rank=000019, location=R00-M0-N08-J35-C00, X-Y-Z-T=03-00-01-0
Index 003:  state=RUNNING         rank=000023, location=R00-M0-N08-J34-C00, X-Y-Z-T=03-01-01-0
Index 004:  state=RUNNING         rank=000022, location=R00-M0-N08-J17-C00, X-Y-Z-T=02-01-01-0
Index 005:  state=RUNNING         rank=000026, location=R00-M0-N08-J18-C00, X-Y-Z-T=02-02-01-0
Index 006:  state=RUNNING         rank=000025, location=R00-M0-N08-J10-C00, X-Y-Z-T=01-02-01-0
Index 007:  state=RUNNING         rank=000029, location=R00-M0-N08-J11-C00, X-Y-Z-T=01-03-01-0
...
```

Table C-2 contains descriptions of each of the fields in Example C-6.

Table C-2 Fields returned with no operand

Field	Description
Index	Identifies the compute node thread in the pset.
state	Current state of the compute node thread.
rank	The MPI rank of the process that the compute node thread is a part of. The value is 0 when the process is not running.
location	The location of the compute node the thread is running on.
X-Y-Z-T	The X-Y-Z-T coordinates of the compute node that the thread is running on.

When a `nodenumber` operand is specified, the output shows detailed information about the compute node thread as show in Example C-7.

Example C-7 Output from show_node_info with operand

```
> show_node_info 5
Index: 5
State: RUNNING
X-Y-Z-T coordinates: X=2 Y=2 Z=1 T=0
Location: R00-M0-N08-J18
Core: 0
Collective address: 26
Process rank: 26
Proxy pid: 910
App leader: true
Message buffer file name: /ciod.rd/ciod-message-buffer-24005
Control message queue: 0
  Number of msgs: 0
  Last send pid:  752
  Last recv pid:  752
Function ship message queue: 294921
  Number of msgs: 0
  Last send pid:  752
  Last recv pid:  785
Debugger message queue: 32769
  Number of msgs: 0
  Last send pid:  0
  Last recv pid:  0
Stdin message queue: 65538
  Number of msgs: 0
  Last send pid:  0
  Last recv pid:  0
Stdout message queue: 98307
  Number of msgs: 0
  Last send pid:  0
  Last recv pid:  0
```

Table C-3 describes the fields shown in Example C-7.

Table C-3 Fields returned with operand

Field	Description
Index	Identifies the compute node thread in the pset.
State	Current state of the compute node thread.
X-Y-Z-T coordinates	The coordinates of the compute node that the thread is running on.
Location	Location of the compute node the thread is running on.
Core	Core number that the thread is running on.
Collective address	Collective address of the compute node the thread is running on.
Process rank	The MPI rank of the process the compute node thread is a part of.
Proxy pid	The process ID of the ioproxy process for the compute node thread.
App leader	True if the compute node thread is responsible for loading the application on the compute node.
Message buffer file name	Path to the file containing the message buffer for the compute node thread.
Number of msgs	The number of messages available on the queue (typically 0).
Last send pid	The process ID of the last process to send a message to the queue
Last recv pid	The process ID of the last process to receive a message from the queue.

For each message queue, the first line has the ID of the message queue. A process Id of 0 means a message has not been sent or received on the queue.

show_process_info

The **`show_process_info`** command shows the list of processes currently running on the compute nodes. There are no operands for the command. Example C-8 displays the output of the **`show_process_info`** command.

Example: C-8 The show_process_info output

```
> show_process_info
Process 000: rank=000007, proxy pid=00924, last signal=-1, tool attached=false, exit reason=0
Process 001: rank=000003, proxy pid=00925, last signal=-1, tool attached=false, exit reason=0
Process 002: rank=000019, proxy pid=00926, last signal=-1, tool attached=false, exit reason=0
Process 003: rank=000023, proxy pid=00932, last signal=-1, tool attached=false, exit reason=0
Process 004: rank=000022, proxy pid=00934, last signal=-1, tool attached=false, exit reason=0
Process 005: rank=000006, proxy pid=00940, last signal=-1, tool attached=false, exit reason=0
Process 006: rank=000002, proxy pid=00941, last signal=-1, tool attached=false, exit reason=0
Process 007: rank=000018, proxy pid=00946, last signal=-1, tool attached=false, exit reason=0
...
```

Each of the fields are defined in Table C-4.

Table C-4 show_process_info fields

Field	Description
Process	Identifies the compute node process in the pset.
rank	The MPI rank of the process.
proxy pid	The process ID of the ioproxy process for the compute node process. The value is -1 when the process is not running.
last signal	The value of the last signal sent to the process. The value is -1 if no signal has been sent.
tool attached	This field shows true if an external tool is attached to the process.
exit reason	Provides the reason the process exited. A value of 0 means the process has not ended yet. A value of 1 means the process ended normally. A value of 2 means the process ended by signal. A value of 3 means the process ended in error.

show_collective_info

The **show_collective_info** command shows detailed information about the collective network. There are no operands for this command. Example C-9 displays the output generated by the **show_collective_info** command.

Example C-9 show_collective_info output.

```
> show_collective_info
Job number: 1
Job id: 445345
Message descriptor thread value: 1
Message descriptor process value: 0
Messages received: 0
Packets received: 0
Packets received in error: 0
Packets sent: 128
Reception fifo data count: 0
Reception fifo header count: 0
Injection fifo data count: 0
Injection fifo header count: 0
Read/write buffer size: 2097152
Max packets per request message: 8194
Max bytes per request message: 2097664
Reply message size: 2097152
Max packets per reply message: 8194
Max bytes per reply message: 2097664
Receive retry count: 50
```

Each of the fields for the show_collective_info command are defined in Table C-5.

Table C-5 Fields for show_collective_info command

Field	Description
Job number	This field is 0 when CIOD starts and increments each time a job is started.
Job id	The job identifier from the control system for the last job started on the block.
Message descriptor thread value	Value for mapping message descriptor to the thread list.
Message descriptor process value	Value for adjusting process when mapping message descriptor to thread list.
Messages received	The total number of messages received by CIOD.
Packets received	The total number of packets received by CIOD.
Packets received in error	The total number of packets received by CIOD that contained an error.
Packets sent	The total number of packets sent by CIOD.
Reception fifo data count	Number of quad-words in the reception data fifo.
Reception fifo header count	Number of headers in the reception header fifo.
Injection fifo data count	Number of quad-words in the injection data fifo.
Injection fifo header count	Number of headers in the injection header fifo.
Read/write buffer size	The size, in bytes, of the CIOD buffer used for read and write operations.
Max packets per request message	Maximum number of packets in request messages sent from compute nodes.
Max bytes per request message	Maximum number of bytes in request messages sent from compute nodes.
Reply message size	Maximum size in bytes of reply messages sent to compute nodes.
Max packets per reply message	Maximum number of packets in reply messages sent to compute nodes.
Max bytes per reply message	Maximum number of bytes in reply messages sent to compute nodes.
Receive retry count	The number of times CIOD tries to receive a packet from the collective network before posting ready messages.

show_collective_addrs

The **show_collective_addrs** command shows detailed information about the collective network configuration. There are no operands for this command. Example C-10 displays the output generated by the **show_collective_addrs** command.

Example C-10 show_collective_addrs output

```
> show_collective_addrs
My address: 8
My location:  R00-M0-N08-J00
Number of threads: 128
Index 000: address=000007, location=R00-M0-N08-J30-C00, X-Y-Z-T=03-01-00-0
Index 001: address=000003, location=R00-M0-N08-J31-C00, X-Y-Z-T=03-00-00-0
Index 002: address=000019, location=R00-M0-N08-J35-C00, X-Y-Z-T=03-00-01-0
Index 003: address=000023, location=R00-M0-N08-J34-C00, X-Y-Z-T=03-01-01-0
Index 004: address=000022, location=R00-M0-N08-J17-C00, X-Y-Z-T=02-01-01-0
Index 005: address=000026, location=R00-M0-N08-J18-C00, X-Y-Z-T=02-02-01-0
Index 006: address=000025, location=R00-M0-N08-J10-C00, X-Y-Z-T=01-02-01-0
Index 007: address=000029, location=R00-M0-N08-J11-C00, X-Y-Z-T=01-03-01-0
...
```

For each compute node thread there is one line that shows the collective address of the compute node the thread is running on, the location of the compute node, and the X-Y-Z-T coordinates of the compute node thread. The other fields for the **show_collective_addrs** command are defined in Table C-6.

Table C-6 Fields for show_collective_addrs

Field	Description
My address	The collective network address for the I/O node
My location	The location of the I/O node
Number of threads	Number of compute node threads being managed by CIOD

show_messages

The **show_messages** command shows detailed information about messages exchanged with compute node threads. If no operand is specified, the default is to show all messages. The the possible operands include:

`all`	Show messages for all compute node threads.
`ready`	Only show messages that are ready.
`posted`	Only show messages that are posted.
`taken`	Only show messages that are taken.
`partial`	Only show messages where part of the packets have been received.
`previous`	Show the previous message for all compute node threads.

The output for the all, ready, posted, taken, and partial types is shown in Example C-11.

Example C-11 show_messages output

```
> show_messages
Index=000, msgpkts=0000, posted=1, taken=1, pktrecv=00012, msgrecv=00011
Index=001, msgpkts=0000, posted=1, taken=1, pktrecv=00012, msgrecv=00011
Index=002, msgpkts=0000, posted=1, taken=1, pktrecv=00012, msgrecv=00011
Index=003, msgpkts=0000, posted=1, taken=1, pktrecv=00012, msgrecv=00011
Index=004, msgpkts=0000, posted=1, taken=1, pktrecv=00012, msgrecv=00011
```

```
Index=005, msgpkts=0000, posted=1, taken=1, pktrecv=00012, msgrecv=00011
Index=006, msgpkts=0000, posted=1, taken=1, pktrecv=00012, msgrecv=00011
Index=007, msgpkts=0000, posted=1, taken=1, pktrecv=00012, msgrecv=00011
```

Each of the fields for the **`show_messages`** command (excluding *previous* type) are defined in Table C-7.

Table C-7 Fields for show_messages

Field	Description
Index	Identifies the compute node thread in the pset.
msgpkts	The current number of packets received for the message.
posted	The posted field is non-zero when the message has been posted for processing by CIOD.
taken	Taken field is non-zero when the message has been taken for processing by CIOD.
pktrecv	The total number of packets received for the compute node thread.
msgrecv	The total number of messages received for the compute node thread.

The output for the previous type is shown in Example C-12.

Example C-12 Output for show_messages previous

```
Index=000, previous protocol=2, previous type=00001, previous size=00256
Index=001, previous protocol=2, previous type=00001, previous size=00256
Index=002, previous protocol=2, previous type=00001, previous size=00256
Index=003, previous protocol=2, previous type=00001, previous size=00256
Index=004, previous protocol=2, previous type=00001, previous size=00256
Index=005, previous protocol=2, previous type=00001, previous size=00256
Index=006, previous protocol=2, previous type=00001, previous size=00256
Index=007, previous protocol=2, previous type=00001, previous size=00256
...
```

Each of the fields for the **`show_messages`** command *previous* are defined in Table C-8.

Table C-8 Fields for show_messages previous

Field	Description
Index	Identifies the compute node thread in the pset.
previous protocol	The protocol number of the previous message.
previous type	The message type of the previous message.
previous size	The number of bytes in the previous message.

show_msgdescriptor_info

The **`show_msgdescriptor_info`** command shows detailed information about a message descriptor. There is one operand for the command, `descriptornumber`, which is the number of the message descriptor. The command uses the following syntax:

```
show_msgdescriptor_info descriptornumber
```

The output of the command is shown in Example C-13.

Example C-13 The show_msgdescriptor_info output

```
> show_msgdescriptor_info 3
Index: 3
Collective address: 23
Core: 0
Total packets received: 2
Total messages received: 2
Control message queue: 0
  Number of msgs: 0
  Last send pid:  759
  Last recv pid:  759
Function ship message queue: 229383
  Number of msgs: 0
  Last send pid:  0
  Last recv pid:  0
Debugger message queue: 32769
  Number of msgs: 0
  Last send pid:  0
  Last recv pid:  0
Stdin message queue: 65538
  Number of msgs: 0
  Last send pid:  0
  Last recv pid:  0
Stdout message queue: 98307
  Number of msgs: 0
  Last send pid:  0
  Last recv pid:  0
Next packet offset: 512
Message packets received: 0
Message posted: 1
Message taken: 1
Terminate: 0
Message protocol: 3
Message type: 67
Message size: 256
Message header: 00430300 00000001 00000000 00000000
Header protocol: 3
Header type: 67
Header size: 0
Header packet total: 1
```

The description of the fields returned by the `show_msgdescriptor_info` are in Table C-9.

Table C-9 Fields for show_msgdescriptor_info

Field	Description
Index	Identifies the message descriptor in the pset.
Collective address	The tree address of the compute node using the message descriptor.
Core	The core number of the compute node thread using the message descriptor.
Total packets received	The total number of packets received from the compute node thread.

Field	Description
Total messages received	The total number of messages received from the compute node thread. For each message queue, the first line has the ID of the message queue,
Number of msgs	The number of messages available on the queue (typically 0).
Last send pid	The process ID of the last process to send a message to the queue. A process ID of 0 means a message has not been sent to the queue.
Last recv pid	The process ID of the last process to receive a message from the queue. A process ID of 0 means a message has not been received on the queue.
Next packet offset	Offset in bytes.
Message packets received	Number of packets received for current message.
Message posted	Value is non-zero when the message has been posted for processing.
Message taken	Value is non-zero when the message has been taken for processing.
Terminate	Value is non-zero when job is terminating
Message protocol	Protocol number of current message.
Message type	Type of current message.
Message size	Number of bytes in current message.
Message header	Hexadecimal dump of header for current message.
Header protocol	Protocol number from message header.
Header type	Type of message from message header.
Header size	Number of bytes in last packet of message from message header.
Header packet total	Number of packets in message from message header.

show_version

The **show_version** command shows the version information. There are no operands for the command. A sample of the output for the command is shown in Example C-14.

Example C-14 The show_version output

```
> show_version
Compile time: 21:35:59 Jan 22 2008
CioStream protocol version: 22
DataStream protocol version: 3
Personality version: 10
```

An explanation of the fields in the output is provided in Table C-10.

Table C-10 Fields for show_version

Field	Description
Compile time	The date and time CIOD was compiled
CioStream protocol version	The version number of the control stream protocol.

Field	Description
Data Stream protocol version	The version number of the data stream protocol.
Personality version	The version number of the Blue Gene/P personality.

show_job_info

The **show_job_info** command shows detailed info about the current job. If no job is current running, the info displayed is from the last job that ran on the block. There are no operands for the command. The output generated by the command is shown in Example C-15.

Example C-15 The show_job_info output

```
> show_job_info
Job number: 1
Job id: 145079
Program name: /bgusr/sam/IOR
Initial working directory: /bgusr/sam
Umask: 0x00000012
User id: 12239 (sam)
Group id: 2633 (sam)
Secondary groups: 2633 (sam) 503 (bgl) 57262 (bgpadmin) 57346 (bgpdeveloper) 57348
(bgpservice) 57347(bgpuser
```

A description of each of the fields returned by the show_job_info is given in Table C-11.

Table C-11 Fields for show_job_info

Field	Description
Job number	The Job number field is 0 when CIOD starts, and increments each time a job is started.
Job id	The job identifier from the control system for the last job started on the block.
Program name	The path to the program for the job.
Initial working directory	The path to the initial working directory for the job.
Umask	The initial umask value for the job.
User id	The numeric user ID and name of the user running the job.
Group id	The primary group ID and name of the user running the job.
Secondary group	List of secondary group ids and names of the user running the job.

show_connection_info

The **show_connection_info** command shows detailed information about the network connections being managed by CIOD. There are no operands for this command. Example C-16 shows the output from the **show_connection_info** command.

Example C-16 The show_connection_info output

```
> show_connection_info
Control stream port: 7000
Control stream is connected to: CioStream socket to 172.16.6.1:47470 using
descriptor 9
Next control stream message: END_MESSAGE (8)
Data stream port: 8000
Data stream is connected to: DataStream socket to 172.16.6.1:57940 using
descriptor 8
Stdin buffer length: 0
Stdin eof received: 0
Service connection port: 9000
Service connection descriptor: 6
Service node IP address: 172.16.6.1
Local IP address: 172.16.103.18
Remote IP address: 172.16.6.1
```

Table C-12 provides a description of the fields returned by the **show_connection_info** command.

Table C-12 Fields for show_connection_info

Field	Description
Control stream port	The TCP port number used for the control connection.
Control stream is connected	Shows the IP address and port number of the peer connected to the control stream.
Next control stream message	The type of the next message expected on the control stream from the control system.
Data stream port	The TCP port number used for the data connection.
Data stream is connected	Shows the IP address and port number of the peer connected to the data stream.
Stdin buffer length	The number of bytes cached in the stdin buffer.
Stdin eof received	The Stdin eof received field is non-zero if end of file has been received for stdin.
Service connection port	TCP port number used for the service connection.
Service connection descriptor	The socket descriptor for the service connection.
Service node IP address	The IP address of the service node
Local IP address	IP address for local side of control system connections.
Remote IP address	IP address for remote sid of control system connections.

show_tool_info

The **show_tool_info** command shows detailed info about an external tool if one is currently running on the IO node. There are no operands for this command. Example C-17 shows the output generated by the **show_tool_info** command.

Example C-17 The show_tool_info output

```
> show_tool_info
Tool read pipe fd: -1
Tool write pipe fd: -1
Tool process id: 0
Debugger thread control: 0
Last debugger timestamp: Mon Mar 10 14:44:45 2008
```

Table C-13 describes the fields returned by the **show_tool_info** command.

Table C-13 Fields for show_tool_info

Field	Description
Tool read pipe fd	Descriptor number of the read pipe connected to the external tool. The value is -1 if no tool is running.
Tool write pipe fd	Descriptor number of the write pipe connected to the external tool. The value is -1 if no external tool is running.
Tool process id	Process ID of the external tool process.
Debugger thread control	The Debugger thread control field is non-zero if the tool thread is allowed to process messages.
Last debugger timestamp	The time the last message was received from the external debugger.

show_config_info

The **show_config_info** command shows the values of the environment variables used to configure CIOD. Example C-18 shows a sample of output from the command. Table 12-3 on page 218 contains descriptions of all of the environment variables used by CIOD.

Example C-18 The show_config_info output

```
> show_config_info
CIOD_RDWR_BUFFER_SIZE: 2097152
CIOD_REPLY_MSG_SIZE: 2097152
CIOD_TREE_RECV_RETRY: 50
CIOD_NEW_JOB_EXIT_PGM: (null)
CIOD_NEW_JOB_EXIT_PGM_TIMEOUT: 3600
CIOD_TRACE_DIRECTORY: (null)
CIOD_LOG_LEVEL: 0
CIOD_END_JOB_TIMEOUT: 4
CIOD_MAX_CONCURRENT_RDWR_OPS: 0
CIOD_MAX_CONCURRENT_LOAD_OPS: 0
CIOD_ENABLE_CONTROL_TRACE: 0
CIOD_MAIN_AFFINITY_MASK: 15 (0xf)
CIOD_READER_AFFINITY_MASK: 15 (0xf)
CIOD_IOPROXY_AFFINITY_MASK: 15 (0xf)
CIOD_READER_INTERRUPT_MODE: 0
CIOD_END_TOOL_TIMEOUT: 10
```

show_sync_info

The **show_sync_info** command shows the state of synchronization objects used by CIOD.There are no operands for the command. The six counter fields show the number of compute node threads in each state. Example C-19 shows the output from the command.

Example C-19 The show_sync_info output

```
> show_sync_info
Reset counter:      0
Load counter:     128
Start counter:      0
Run counter:        0
End counter:        0
Finished counter:   0

Debugger thread control: 0
Concurrent rd/wr operations: 0
Concurrent load operations: 0

Control message queue: 0
  Number of msgs: 0
  Last send pid:  737
  Last recv pid:  737
Debugger message queue: 32769
  Number of msgs: 0
  Last send pid:  0
  Last recv pid:  0
Stdin message queue: 65538
  Number of msgs: 0
  Last send pid:  0
  Last recv pid:  0
Stdout message queue: 98307
  Number of msgs: 0
  Last send pid:  0
  Last recv pid:  0
```

For each message queue, the first line has the ID of the message queue. The fields for the **show_sync_info** are described in Table C-14.

Table C-14 Fields for show_sync_info

Field	Description
Debugger thread control	Value is non-zero if the debugger thread is allowed to process messages.
Concurrent rd/wr operations	Value is non-zero when there are concurrent I/O operations in progress.
Concurrent load operations	Value is non-zero when there are concurrent load operations in progress.
Number of msgs	The number of messages available on the queue (typically 0),

Field	Description
Last sent pid	The process ID of the last process to send a message to the queue. A process ID of 0 means a message has not been sent to the queue.
Last recv pid	The process ID of the last process to receive a message from the queue. A process ID of 0 means a message has not been received on the queue.

tracedir

The **tracedir** command sets the directory prefix for trace files. The command uses the following syntax:

```
tracedir directory
```

The directory is the path to the directory prefix for the trace files. When tracing is turned on with the **tracedir** command, trace data is output to a file that is created in directory. CIOD creates a subdirectory in directory using the location string for the I/O node. Any trace data generated by the main CIOD process is stored in a file named ciod.trace.*nnn* where *nnn* is the target number. Any trace data generated by the ioproxy processes is stored in a file named ioproxy.trace.*nnn* where *nnn* is the process index.

trace

The **trace** command turns trace on or off for a category of trace points. The command uses the following syntax:

```
trace <category> on|off
```

Categories for the trace command include:

CONTROL	Traces control flow events.
DATA	Traces data stream events.
DEBUGGER	Traces debugger events.
FILEIO	Traces file I/O events.
INFO	Traces extra informational events.
TEST	Current unused.

The trace data is output on the service connection and optionally to a file, except for FILEIO, which can be output only to a file. The trace data shows detailed information about the processing that CIOD performs.

show_trace_info

The **show_trace_info** command shows detailed information about trace. There are no operands for this command. Example C-20 shows the output from the command.

Example C-20 The show_trace_info output

```
> show_trace_info
Trace file name: /bgusr/sam/R00-M0-N08-J00/ciod.trace.000
Trace file fd: 8
CONTROL   on
DATA      off
DEBUGGER  off
FILEIO    off
INFO      off
TEST      off
```

Table C-15 describes the fields that display in the **`show_trace_info`** command output.

Table C-15 Fields for show_trace_info

Field	**Description**
Trace file name	The path to the current trace file
Trace file fd	The descriptor number for the current trace file

The remainder of the fields show if the trace category is on or off.

loglevel

The **`loglevel`** command controls the amount of data written by CIOD to the I/O node log file. The command uses the following syntax:

```
loglevel [level]
```

The following level values are allowed:

level 0	CIOD does no additional logging.
level 1	CIOD logs additional information for recoverable errors.
level 2	CIOD also logs when jobs start and end.
level 3	CIOD also logs performance data for selected system calls.

The log level is accumulative so that when the level is 3 the information for log levels 1 and 2 is also written to the log file. If level is not specified, the current log level is displayed.

D

High Availability service node

Having one service node that manages the stateless Blue Gene/P hardware makes many administrative tasks very simple. However, the single Blue Gene service node was considered a single point of failure. When a service node fails, users can no longer run Blue Gene applications, and the system administrator cannot monitor, control, and maintain the Blue Gene system.

Beginning with V1R3M0, you can use the Blue Gene/P system in a high availability configuration that consists of a primary service node and a secondary service node.

To detect service node failures and to fail over automatically to the backup service node, we recommend a solution using Linux-HA (which is shipped with most Linux distributions, with the exception of Red Hat®). Linux-HA allows for the management of unplanned and planned outages.

From the hardware perspective, we recommend that the service node be made highly available by adding a backup service node, a tie-breaker node (possibly use a front end node) and external storage for /bgsys and /dbhome. The backup service node capacity and performance does not need to be identical to the primary service node. The decision on the backup service node depends on the Blue Gene system requirements when the primary service node is not available.

One of the challenges that can arise in an HA environment is known as *split-brain*, where due to a communication problem, both service nodes attempt to take over the HA resources. To avoid the issue of split-brain, we recommend using a front end node as a quorum server. If both the primary and the secondary servers are trying to take over resources, the front end node can provide an extra quorum vote. Because the quorum server is not a cluster member, it cannot take over the resources.

We have identified several key components that, along with the service node hardware, must be considered in a failover situation:

- The `/bgsys` file system
- The `/dbhome` file system
- BGPMaster
- NFS
- Other file servers
- Job schedulers

The `/bgsys` file system contains the Blue Gene/P driver code, and `/dbhome` is the home directory for the Blue Gene/P DB2 database. Using external storage for `/bgsys` and `/dbhome` provides for continuous access to the Blue Gene/P software and database (configuration, logs, RAS, and environmental data) after the failure of the active service node. Using external storage ensures that Blue Gene control programs, user applications, and DB2 databases can continue to function with little impact on the jobs when running on the backup service node.

Obviously, any jobs running on the system at the time of the active service node failure are lost. However, after the service node failover process completes, users can restart failed jobs. Job schedulers, such as LoadLeveler, can be set to resubmit a job in cases where a failover is detected.

You can find an example of how to configure an HA environment in the Blue Gene Knowledge Base at:

`https://www-912.ibm.com/i_dir/IBMBlueGeneLKB.nsf/wall`

You are prompted for your IBM ID and password. After you log in to the Knowledge Base, search for document *504319661*.

E

Setting up the cross-compile environment

When building an application in a cross-compile environment such as Blue Gene/P, build tools, such as **`configure`** and **`make`**, will sometimes compile and execute small code snippets to identify characteristics of the target platform as part of the build process. If these code snippets are compiled with a cross-compiler and then executed on the build system instead of the target system, the program might fail to execute or might produce results that do not reflect the target system. When this situation occurs, the configuration fails or does not configure as expected.

To avoid this problem, starting at V1R4M0, Blue Gene/P provides a method to run Blue Gene/P executables transparently on a Blue Gene/P partition when executed on a Blue Gene/P front end node as though they are being executed natively. Because these programs execute and produce the appropriate results for the target system, the **`configure`** and **`make`** commands run correctly and return the expected results for the target system.

This feature provides code in the program loader on the front end node that verifies whether the program was compiled for Blue Gene/P. If so, rather than executing the program, it invokes a **`submit`** command with the same program and arguments to run the program on the Blue Gene/P partition. In addition to remote program execution, all input and output from the program is handled transparently, including stdin, stdout, stderr, arguments, return codes, and signals.

For developers to use this feature, a Blue Gene/P partition must be made available. Typically, the Blue Gene/P system is controlled by a resource scheduler. In this case, you might be required to configure the scheduler to not schedule jobs to run on the partition to be used for configuring applications.

The selected partition must be booted in HTC mode using the following command:

```
$ /bgsys/drivers/ppcfloor/bin/htcpartition --boot --partition <partition id>
```

The front end nodes must be set up by copying the `/bgsys/drivers/ppcfloor/lib/ld-bg-launcher.so` file to `/lib`, as shown in the following command, on each front end node as root:

```
# cp /bgsys/drivers/ppcfloor/lib/ld-bg-launcher.so /lib
```

The environment for the developers must also be modified to take advantage of this feature. You can create a file in `/etc/profile.d` on each front end node with the contents shown in Example E-1.

Example E-1 profile.d contents

```
export BG_PGM_LAUNCHER=submit
export HTC_SUBMIT_POOL=<partition id>
```

Now the system is ready to be used.

F

Setting up the Event Log Analysis Environment

The Event Log Analysis (ELA) utility requires that PyDB2 V1.1 and Python V2.6 (64-bit) are installed on the service node. This appendix provides instructions for installing the additional products.

Note: At the time of this writing the current version of Python is V2.6.0-2.14 and PyDB2 is V1.1.1-1. If you have a different version, modify the instructions in this appendix accordingly.

You can download the PyDB2 source from the SourceForge Web site:

`http://sourceforge.net/projects/pydb2/files/`

You can download Python from the openSUSE.org Web site:

`http://download.opensuse.org/source/distribution/11.1/repo/oss/suse/src/python-base-2.6.0-2.14.src.rpm`

Build and install Python-base-2.6

Note: Instructions for compiling a version 2.6 64-bit Python interpreter from source and relocating it to /usr/local so it will not conflict with the existing SLES 2.4-supplied Python RPMs are provided in Blue Gene Knowledge Base document 534527039.

The following instructions will build these RPMs:

- `python-base`
- `python-xml`
- `python-devel`
- `libpython`

This process provides:

`python-base`	`/usr/bin/pydoc`, `/usr/bin/python` 64-bit interpreters, and `/usr/lib64` Python libraries
`python-xml`	`/usr/lib64/python2.6/xml libraries`
`python-devel`	`/usr/include/python2.6`, test files, and `python-config`
`libpython`	`/usr/lib64/libpython2.6.so.1.0`

To build and install Python-base, follow these steps:

1. Download the Python-base source RPM (signed in as root):

```
# cd /tmp
# wget
http://download.opensuse.org/source/distribution/11.1/repo/oss/suse/src/pyth
on-base-2.6.0-2.14.src.rpm
# rpm -ivh /tmp/python-base-2.6.0-2.14.src.rpm
```

2. Install the build prerequisites:

```
# yast -i blt openssl-devel openssl-devel-64bit readline-devel sqlite-devel
tk-devel sqlite-64bit readline-devel-64bit readline-64bit
```

Note: The **`yast -i`** command works only if you have set up YaST installation sources previously using **`yast inst_source`**.

3. Prepare the environment to build the RPM:

```
# export ARCH=ppc64
# export CFLAGS="-m64"
# export LDFLAGS="-m64"
# export LIBPATH=$LIBPATH:/usr/lib64
# unset DISPLAY
# cd /usr/lib64
# ln -s libsqlite3.so.0 libsqlite3.a
```

4. Modify the RPM source files:

```
# cd /tmp/
# tar -xvjf /usr/src/packages/SOURCES/Python-2.6.tar.bz2
# cd Python-2.6
```

   a. Edit the `configure.in file`, and change `AC_PREREQ(2.61)` to `AC_PREREQ(2.59)`.

   b. Edit the `setup.py` file. At around line 664, look for the following clause:

```
if (ssl_incs is not None and
    ssl_libs is not None and
 openssl_ver >= 0x00907000):
```

At the bottom of this clause, just before the else clause, add the following lines:

```
exts.append( Extension('_md5',
            sources = ['md5module.c', 'md5.c'],
            depends = ['md5.h']) )
exts.append( Extension('_sha256', ['sha256module.c']) )
exts.append( Extension('_sha512', ['sha512module.c']) )
exts.append( Extension('_sha', ['shamodule.c']) )
```

c. Run the following commands (as root):

```
# rm /tmp/Python-2.6/Lib/test/test_httpservers.py
# cd /tmp
# tar -cvjf Python-2.6.tar.bz2 Python-2.6
# cp Python-2.6.tar.bz2 /usr/src/packages/SOURCES/
```

5. Modify the SPEC file:

```
# vi /usr/src/packages/SPECS/python-base.spec
```

Remove the line `--docdir=%{_docdir}/python \` at line 154. It is an unsupported argument to `autoconf 2.59`.

Add the following lines to the SPEC file in the required `spec` section, which should begin after line 363:

```
%{_libdir}/python%{python_version}/lib-dynload/_bsddb.so
%{_libdir}/python%{python_version}/lib-dynload/_curses.so
%{_libdir}/python%{python_version}/lib-dynload/_curses_panel.so
%{_libdir}/python%{python_version}/lib-dynload/_hashlib.so
%{_libdir}/python%{python_version}/lib-dynload/_sqlite3.so
%{_libdir}/python%{python_version}/lib-dynload/_ssl.so
%{_libdir}/python%{python_version}/lib-dynload/_tkinter.so
%{_libdir}/python%{python_version}/lib-dynload/dbm.so
%{_libdir}/python%{python_version}/lib-dynload/gdbm.so
%{_libdir}/python%{python_version}/lib-dynload/readline.so
```

6. Build the RPMs:

```
rpmbuild -v -bb /usr/src/packages/SPECS/python-base.spec > /tmp/build.log 2>&1
```

The final RPMs are located in `/usr/src/packages/RPMS/ppc64`.

7. Install the RPMs (all one command):

Suggestion: Before installing the programs you can set permissions using the umask command:

```
umask 022
```

```
rpm -ivh --force
/usr/src/packages/RPMS/ppc64/libpython2_6-1_0-2.6.0-2.14.ppc64.rpm
/usr/src/packages/RPMS/ppc64/python-base-2.6.0-2.14.ppc64.rpm
/usr/src/packages/RPMS/ppc64/python-devel-2.6.0-2.14.ppc64.rpm
/usr/src/packages/RPMS/ppc64/python-xml-2.6.0-2.14.ppc64.rpm
```

Installing PyDB2

Because you must compile PyDB2 on a system that has DB2 installed, we used our service node to complete the following steps:

1. Sign in as root, and download the PyDB2 `.tar` file:

   ```
   # cd /tmp
        # wget
   http://sourceforge.net/projects/pydb2/files/pydb2/PyDB2_1.1.1-1/PyDB2_1.1.1-
   1.tar.gz/download
   ```

2. Extract the file and modify the setup file:

   ```
   # tar -zxvf PyDB2_1.1.1-1.tar.gz
   # cd PyDB2_1.1.1
   ```

   a. Modify the `DB2_ROOT` entry in the setup file to match your DB2 installation directory:

      ```
      # vi setup.py
      ```

      Depending on the version installed, you can have the following possible values:

      - `DB2_ROOT = "/opt/ibm/db2/V9.1/"`
      - `DB2_ROOT = "/opt/ibm/db2/V9.5/"`

   b. Change the library path on line 82 from:

      ```
      for path in ['lib', 'lib32', 'lib64']:
      ```

      to:

      ```
      for path in ['lib', 'lib64']:
      ```

3. Prepare the installation environment:

   ```
   # export CFLAGS="-m64"
   ```

4. Source the DB2 profile:

   ```
   # source /dbhome/bgpsysdb/sqllib/db2profile
   ```

5. Install PyDB2:

   ```
   # python setup.py install --prefix=/usr
   ```

Verifying the installation

Use these steps to verify that Python is installed correctly.

1. Verify the install directory using the following command:

   ```
   > which python
   ```

   This command should return the following directory:

   ```
   /usr/bin/python
   ```

2. Initiate a Python session using the **`python`** command (see Example F-1) and issue the following commands:

   a. Type the command **`import DB2`**, which should return silently to a new prompt.

   b. Type the command **`import platform`**, which should also return silently to a new prompt.

   c. Type the command **`platform.architecture()`**, which should return the following line:

```
    ('64bit', 'ELF')
```

Example F-1   Python session

```
bgpadmin@yourSN:~> python
Python 2.6 (r26:66714, Aug 21 2009, 12:20:36)
[GCC 4.1.2 20070115 (SUSE Linux)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import DB2
>>> import platform
>>> platform.architecture()
('64bit', 'ELF')
>>>
```

3. Then, press a Ctrl-d to exit the Python session.

If there are no errors returned during this verification, Python and PyDB2 are installed correctly.

Glossary

32b executable Executable binaries (user applications) with 32b (4B) virtual memory addressing. Note that this is independent of the number of bytes (4 our 8) utilized for floating-point number representation and arithmetic.

32b floating-point arithmetic Executable binaries (user applications) with 32b (4B) floating-point number representation and arithmetic. Note that this is independent of the number of bytes (4 our 8) utilized for memory reference addressing.

32b virtual memory addressing All virtual memory addresses in a user application are 32b (4B) integers. Note that this is independent of the type of floating-point number representation and arithmetic.

64b executable Executable binaries (user applications) with 64b (8B) virtual memory addressing. Note that this is independent of the number of bytes (4 our 8) utilized for floating-point number representation and arithmetic. Note that all user applications should be compiled, loaded with Subcontractor supplied libraries and executed with 64b virtual memory addressing by default..

64b floating-point arithmetic Executable binaries (user applications) with 64b (8B) floating-point number representation and arithmetic. Note that this is independent of the number of bytes (4 our 8) utilized for memory reference addressing.

64b virtual memory addressing All virtual memory addresses in a user application are 64b (8B) integers. Note that this is independent of the type of floating-point number representation and arithmetic. Note that all user applications should be compiled, loaded with Subcontractor supplied libraries and executed with 64b virtual memory addressing by default.

AICF The Argonne Leadership Computing Facility operated for the U.S. Department of Energy's Office of Science by UChicago Argonne LLC.

API Application Programming Interface. Defines the syntax and semantics for invoking services from within an executing application. All APIs shall be available to both Fortran and C programs, although implementation issues (such as whether the Fortran routines are simply wrappers for calling C routines) are up to the supplier**.**

ASIC Application Specific Integrated Circuit.

ATP Acceptance Test Plan

b bit. A single, indivisible binary unit of electronic information.

B byte. A collection of eight (8) bits.

BGP Blue Gene/P. The name given to the collection of compute nodes, I/O nodes, front end nodes, file systems, and interconnecting networks that is the subject of this statement of work.

BGPEA BGP Early Access platform

BPC BGP Compute ASIC. This high-function BGP ASCI is the basis of the compute nodes and I/O nodes.

BPL BGP Link ASIC. This high-function BGP ASCI is responsible for redriving communications signals between midplanes and is used to repartition BGP

CE On-site hardware customer engineer performing hardware maintenance.

cluster A set of nodes connected through a scalable network technology**.**

CMCS Cluster Monitoring and Control System.

CMN Control and Management Network. Provides a command and control path to BGP for functions such as, but not limited to, health status monitoring, repartitioning, and booting.

CN compute node. The element of BGP that supplies the primary computational resource for execution of a user application.

compute card One of the field replaceable units (FRUs) of BGP. A compute card contains two complete compute nodes, and is plugged into a node card.

compute node (CN) The element of BGP that supplies the primary computational resource for execution of a user application.

Core The BGP Core is Subcontractor delivered hardware and software. The BGP Core consists of the BGP Compute Main Section, front-end node (FEN), service node (SN), and a control and management Ethernet.

CPU Central Processing Unit or processor. A VLSI chip constituting the computational core (integer, floating point, and branch units), registers and memory interface (virtual memory translation, TLB and bus controller).

current standard Applied when an API is not *frozen* on a particular version of a standard, but shall be upgraded automatically by the Subcontractor as new specifications are released. For example, *MPI version 2.0* refers to the standard in effect at the time of writing this document, while *current version of MPI* refers to further versions that take effect during the lifetime of this contract.

CWFS Cluster Wide File System. The file system that is visible from every node in the system with scalable performance.

DDR Double Data Rate. A technique for doubling the switching rate of a circuit by triggering on both the rising edge and falling edge of a clock signal.

EDRAM Enhanced dynamic random access memory is dynamic random access memory that includes a small amount of static RAM (SRAM) inside a larger amount of DRAM. Performance is enhanced by organizing so that many memory accesses will be to the faster SRAM.

FEN Front-End Node. The front-end node is responsible, in part, for interactive access to Blue Gene/P.

FGES Federated Gigabit-Ethernet Switch. The FGES connects the I/O nodes of Blue Gene/P to external resources, such as the FEN and the CWFS.

FLOP or OP Floating Point Operation

FLOPS or OPS Plural of FLOP

FLOPs or OPs Floating Point Operations per second.

front-end node The front-end node (FEN) is responsible, in part, for interactive access to Blue Gene/P

FRU Field Replaceable Unit.

fully supported Refers to product-quality implementation, documented and maintained by the HPC machine supplier or an affiliated software supplier.

GB gigabyte. A billion base 10 bytes. This is typically used in every context except for Random Access Memory size and is 109 (or 1,000,000,000) bytes.

GiB gibibyte. A billion base 2 bytes. This is typically used in terms of Random Access Memory and is 230 (or 1,073,741,824) bytes.

GFlops or GOPs gigaflops. A gigaflops is a billion (109 = 1,000,000,000) 64-bit floating point operations per second.

host complex The host complex includes the front-end node (FEN) and service node (SN).

HSN Hot Spare Node.

ICON The ICON is a high-function BGP ASIC that is responsible for Ethernet-to-JTAG conversion and other control functions.

IP The Internet Protocol (IP) is the method by which data is sent from one computer to another on the Internet.

job A cluster wide abstraction similar to a POSIX session, with certain characteristics and attributes. Commands shall be available to manipulate a job as a single entity (including kill, modify, query characteristics, and query state).

I/O I/O (input/output) describes any operation, program, or device that transfers data to or from a computer.

I/O card One of the field replaceable units (FRUs) of BGP. An I/O card contains two complete I/O nodes, and is plugged into a node card.

I/O Node The I/O nodes (ION) are responsible, in part, for providing I/O services to compute nodes.

IBM International Business Machines Corporation.

ION I/O Node. Responsible, in part, for providing I/O services to compute nodes.

LINPACK LINPACK is a collection of Fortran subroutines that analyze and solve linear equations and linear least-squares problems.

Linux from Linus (Torvalds; program author's name) and UNIX®. Linux is a free UNIX-like operating system originally created by Linus Torvalds with the assistance of developers around the world. Developed under the GNU General Public License, the source code for Linux is freely available to everyone.

MB megabyte. A million base 10 bytes. Typically used in every context except for Random Access Memory size and is 106 (or 1,000,000) bytes.

MiB mebibyte. A million base 2 bytes. Typically used in terms of Random Access Memory and is 220 (or 1,048,576) bytes. For a complete description of SI units for prefixing binary multiples see:
`http://physics.nist.gov/cuu/Units/binary.html`

midplane An intermediate packaging component of BGP. Multiple node cards plug into a midplane to form the basic scalable unit of BGP.

MFlops or MOPs megaflops. A million (106 = 1,000,000) 64-bit floating point operations per second.

MPI Message Passing Interface.

MPICH2 MPICH is an implementation of the MPI standard available from Argonne National Laboratory.

MTBF Mean Time Between Failure. MTBF is a measurement of the expected reliability of the system or component. The MTBF figure can be developed as the result of intensive testing, based on actual product experience, or predicted by analyzing known factors. See: `http://www.t-cubed.com/faq_mtbf.htm`

node A node operates under a single instance of an operating-system image, and is an independent operating-system partition.

node card An intermediate packaging component of BGP. FRUs (field-replaceable units - compute cards and I/O cards) are plugged into a node card. Multiple node cards plug into a midplane to form the basic scalable unit of BGP.

OpenMP OpenMP is a portable, scalable model that gives shared-memory parallel programmers a simple and flexible interface for developing parallel applications.

peak rate The peak rate is the maximum number of 64-bit floating point instructions (add, subtract, multiply or divide) per second that could conceivably be retired by the system. For RISC CPUs the peak rate is typically calculated as the maximum number of floating point instructions retired per clock times the clock rate.

PTRACE A facility that allows a parent process to control the execution of a child process. Its primary use is for the implementation of breakpoint debugging.

published (as applied to APIs) This term shall refer to the situation where an API is not required to be consistent across platforms. An "published" API shall refer to the fact that the API shall be documented and supported, although it shall be Subcontractor, or even platform specific.

RAID Redundant Array of Independent (Inexpensive) Disks.

RAM Random Access Memory.

RAS Reliability, Availability and Serviceability.

SAN Storage Area Network, a high-speed subnetwork of storage devices.

scalable A system attribute that increases in performance or size as some function of the peak rating of the system. The scaling regime of interest is at least within the range of 1 teraflops to 60.0 (and possibly to 120.0) teraFLOPs peak rate.

service node The service node is responsible, in part, for management and control of Blue Gene/P.

SDRAM Synchronous, Dynamic Random Access Memory

single-point control (as applied to tool interfaces) The ability to control or acquire information about all processes/PEs using a single command or operation.

SMFS System Management File System. Provides a single, central location for administrative information about BGP.

SMP Symmetric Multi-Processor. A computing node in which multiple functional units operate under the control of a single operating-system image.

SN Service Node. Responsible, in part, for management and control of Blue Gene/P.

SPMD Single Program Multiple Data. SPMD is a programming model wherein multiple instances of a single program operates on multiple data.

sPPM The sPPM benchmark solves a 3D gas dynamics problem on a uniform Cartesian mesh, using a simplified version of the PPM (Piecewise Parabolic Method) code.

SRAM Static Random Access Memory.

standard (as applied to APIs) Where an API is required to be consistent across platforms, the reference standard is named as part of the capability. The implementation shall include all routines defined by that standard (even if some simply result in no-ops on a given platform).

TB terabyte. A terabyte is a trillion base 10 bytes. This is typically used in every context except for Random Access Memory size and is 1012 (or 1,000,000,000,000) bytes.

TCP/IP Transmission Control Protocol/Internet Protocol, the suite of communications protocols used to connect hosts on the Internet.

TiB tebibyte. A trillion bytes base 2 bytes. This is typically used in terms of Random Access Memory and is 240 (or 1,099,511,627,776) bytes. For a complete description of SI units for prefixing binary multiples see: `http://physics.nist.gov/cuu/Units/binary.html`

TFlops teraflops. A trillion (1012 = 1,000,000,000,000) 64-bit floating point operations per second.

UMT2000 The UMT benchmark is a 3D, deterministic, multigroup, photon transport code for unstructured meshes.

UPC Unified Parallel C, a programming language with parallel extensions to ANSI C. See: `http://upc.gwu.edu/`

XXX-compatible (as applied to system software and tool definitions) This term shall require that a capability be compatible, at the interface level, with the referenced standard, although the lower-level implementation details will differ substantially. For example, "NFSv4-compatible" means that the distributed file system shall be capable of handling standard NFSv4 requests, but need not conform to NFSv4 implementation specifics.

Related publications

We consider the publications that we list in this section particularly suitable for a more detailed discussion of the topics that we cover in this book.

IBM Redbooks publications

For information about ordering these publications, see “How to get IBM Redbooks publications” on page 279. Note that some of the documents referenced here might be available in softcopy only.

Blue Gene/L applications:

- *Unfolding the IBM eServer Blue Gene Solution*, SG24-6686
- *Blue Gene/L: Performance Analysis Tools*, SG24-7278
- *IBM System Blue Gene Solution: Application Development*, SG24-7179

Hardware, Software, and System Administration:

- *IBM System Blue Gene Solution: System Administration*, SG24-7178
- *IBM System Blue Gene Solution: Configuring and Maintaining Your Environment*, SG24-7352
- *IBM System Blue Gene Solution: Hardware Installation and Serviceability*, SG24-6743
- *IBM System Blue Gene Solution: Blue Gene/P Safety Considerations*, REDP-4257
- *Blue Gene/L: Hardware Overview and Planning*, SG24-6796
- *GPFS Multicluster with the IBM System Blue Gene Solution and eHPS Clusters*, REDP-4168
- *IBM System Blue Gene Solution Problem Determination Guide*, SG24-7211
- *Evolution of the IBM System Blue Gene Solution*, REDP-4247

How to get IBM Redbooks publications

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

To determine the spine width of a book, you divide the paper PPI into the number of pages in the book. An example is a 250 page book using Plainfield opaque 50# smooth which has a PPI of 526. Divided 250 by 526 which equals a spine width of .4752". In this case, you would use the .5” spine. Now select the Spine width for the book and hide the others: **Special>Conditional Text>Show/Hide>SpineSize(-->Hide:)>Set** . Move the changed Conditional text settings to all files in your book by opening the book file with the spine.fm still open and **File>Import>Formats** the Conditional Text Settings (ONLY!) to the book files.

IBM Redbooks

IBM System Blue Gene Solution: Blue Gene/P System Administration

(1.5” spine)
1.5”<-> 1.998”
789 <->1051 pages

IBM Redbooks

IBM System Blue Gene Solution: Blue Gene/P System Administration

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

IBM Redbooks

IBM System Blue Gene Solution: Blue Gene/P System Administration

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

IBM Redbooks

IBM System Blue Gene Solution: Blue Gene/P System Administration

(0.2”spine)
0.17”<->0.473”
90<->249 pages

(0.1”spine)
0.1”<->0.169”
53<->89 pages

To determine the spine width of a book, you divide the paper PPI into the number of pages in the book. An example is a 250 page book using Plainfield opaque 50# smooth which has a PPI of 526. Divided 250 by 526 which equals a spine width of .4752". In this case, you would use the .5” spine. Now select the Spine width for the book and hide the others: **Special>Conditional Text>Show/Hide>SpineSize(-->Hide:)>Set** . Move the changed Conditional text settings to all files in your book by opening the book file with the spine.fm still open and **File>Import>Formats** the Conditional Text Settings (ONLY!) to the book files.

IBM

Redbooks

IBM System Blue Gene Solution: Blue Gene/P System Administration

(2.5” spine)
2.5”<->nnn.n”
1315<-> nnnn pages

IBM

Redbooks

IBM System Blue Gene Solution: Blue Gene/P System Administration

(2.0” spine)
2.0” <-> 2.498”
1052 <-> 1314 pages

IBM System Blue Gene Solution: Blue Gene/P System Administration

Perform Blue Gene/P system administration

Configure and use Blue Gene Navigator

Run Diagnostics and perform Service Actions

This IBM Redbooks publication is one in a series of books that are written specifically for the IBM System Blue Gene supercomputer, Blue Gene/P, which is the second generation of massively parallel supercomputer from IBM in the Blue Gene series. It provides an overview of the system administration environment for Blue Gene/P. It is intended to help administrators understand the tools that are available to maintain this system.

This book explains briefly the evolution of the system, from Blue Gene/L to the new Blue Gene/P. It details Blue Gene Navigator, which has grown to be a full featured Web-based system administration tool on Blue Gene/P.

The book also describes many of the day-to-day administrative functions, such as running Diagnostics, performing Service Actions, and monitoring hardware. There are also sections to cover BGPMaster and the processes that it monitors. The final chapters of the book cover the tools that are shipped with the system and details about configuring communications with the I/O nodes and Power Management features.

SG24-7417-03 ISBN 0738433357